TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

PERTTU PAARLAHTI

UUDELLEENKÄYTETTÄVÄN KONEENOHJAUSJÄRJESTELMÄN SUUNNITTELU JA TOTEUTUS

Diplomityö

Tarkastaja: professori Jukka Vanhala
Tarkastaja ja aihe hyväksytty
30. marraskuuta 2017

TIIVISTELMÄ

PERTTU PAARLAHTI: Uudelleenkäytettävän koneenohjausjärjestelmän suunnittelu ja toteutus
Tampereen teknillinen yliopisto
Diplomityö, 83 sivua, 1 liitesivu
Elokuu 2018
Sähkötekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Sulautetut järjestelmät
Tarkastaja: professori Jukka Vanhala

Avainsanat: koneenohjausjärjestelmät, sulautetut järjestelmät, ohjelmistoarkkitehtuuri, ohjelmiston uudelleenkäyttö

Koneenohjausjärjestelmä saa koneen toimimaan turvallisesti ja operaattorin komentojen mukaisesti. Yhä suurempi osa koneenohjausjärjestelmien toiminnallisuudesta toteutetaan nykyisin ohjelmistolla. Koneenohjausjärjestelmän ohjelmiston suorituksesta vastaa yksi tai useampi ohjelmoitava ohjausyksikkö. Ohjausyksiköillä voi olla lukuisia erilaisia IO-liitäntöjä, joilla ne liittyvät antureihinsa ja toimilaitteisiinsa. Ohjausyksiköt liittyvät toisiinsa kommunikaatioväylien avulla. Väylätoteutuksia on olemassa useita, ja kullakin väylätoteutuksella voi olla useita korkean tason protokollia. Koneet ovat väärin toimiessaan vaarallisia, minkä vuoksi koneiden turvallisuudelle on asetettu erityisiä vaatimuksia laissa ja sovellusalakohtaisissa standardeissa.

Koneenohjausjärjestelmien ohjaus toteutettiin ennen riittävän edullisten ja kestävien ohjausyksiköiden kehittymistä releillä ja ajastimilla, mikä tarjosi hyvin vähän joustovaraa vaatimusten muuttamiselle. Ensimmäiset ohjelmoitavat ohjausyksiköt ohjelmoitiin rajallisten laitteistoresurssien vuoksi hyvin matalan tason ohjelmointikielillä ja proseduraalista ohjelmointiparadigmaa noudattaen. Ohjausyksikköjen suorituskyvyn kehittyminen on mahdollistanut korkean tason ohjelmointikielten käyttämisen ja olio-ohjelmoinnin soveltamisen.

Koneenohjausjärjestelmien ohjelmiston vaatimusten kasvaessa ohjelmiston uudelleenkäytöstä on tullut tärkeää. Uudelleenkäyttöä vaikeuttavat ohjelmiston laitteistoläheisyys ja vaihtelevat ominaisuudet. Ilman tehokasta ja systemaattista ohjelmiston uudelleenkäyttöä koneenvalmistaja ei pysty kilpailemaan koneiden kohtuullisella hinnalla ja uusimmalla teknologialla. Olio-ohjelmointikielten yleistyminen sulautetuissa järjestelmissä parantaa ohjelmiston uudelleenkäytettävyyttä. Olio-ohjelmoinnin käyttö itsessään ei kuitenkaan takaa uudelleenkäytettävyyttä, vaan ohjelmisto on suunniteltava huolellisesti erityisesti uudelleenkäyttöä varten. Olio-ohjelmoinnin SOLID-periaatteet auttavat uudelleenkäytettävän ja helposti ymmärrettävän, jatkokehitettävän ja ylläpidettävän ohjelmiston suunnittelussa.

Olen viimeisen vuoden aikana kehittänyt Sandvikille uutta koneenohjausjärjestelmää kaivoslastaus- ja -kuljetuskoneille. Järjestelmän on tarkoitus olla uudelleenkäytettävä laajalle kirjolle erilaisia koneita ja konemalleja tulevaisuudessa. Tässä diplomityössä kuvataan, miten olio-ohjelmointia, uudelleenkäytettävän ohjelmiston suunnitteluperiaatteita ja muita koneenohjausjärjestelmien yleisiä suunnittelumalleja on sovellettu käytännössä. Vaikka uudelleenkäytettävyys on otettu suunnittelussa huomioon, vasta järjestelmän käyttöönotto tulevaisuudessa paljastaa todellisen uudelleenkäytettävyyden.

ABSTRACT

PERTTU PAARLAHTI: Design and implementation of a reusable machine control system
Tampere University of Technology
Master of Science Thesis, 83 pages, 1 Appendix page
August 2018
Master's Degree Programme in Electrical Engineering
Major: Embedded systems
Examiner: Professor Jukka Vanhala

Keywords: machine control systems, embedded systems, software architecture, software reuse

Machine control system makes a machine operate safely and according to the operator's commands. Nowadays ever-increasing portion of the system functionality is implemented by software. Control system's software is run by one or multiple programmable control units. Control units may have various kinds of IO-connections with its sensors and actuators. Control units are interconnected with communication buses. There is a vast variety of communication bus implementations, and each bus implementation may have multiple different high-level protocols available. Malfunctioning machines are dangerous, and therefore legislators and domain specific standards have set specific requirements for machine safety.

Prior to existence of cheap and durable enough control units, machine control systems were implemented connecting relays and cam timers. There was very little flexibility for changing system requirements. Due to limited hardware resources, early programmable control units were programmed with very low-level programming languages and using procedural programming paradigm. As hardware capabilities of control units improved, using high-level programming languages and object-oriented programming paradigm became viable options.

Software reuse has become urgent as machine control system software requirements have increased over time. Hardware dependency and varying requirements are making machine control system software reuse difficult. Without efficient and systematic software reuse, machine manufacturers are not able to compete with reasonable prices and latest technologies. Object-oriented programming becoming more wide-spread in embedded software has improved software reusability. Object-oriented programming by itself does not guarantee reusability. The system still needs to be carefully designed especially for reusability. Following the SOLID-principles for object-oriented programming helps designing the system to be reusable and easy to understand, maintain and develop further.

I have been developing a new machine control system for loading and hauling machines for Sandvik during the last year. The new control system is intended to be reusable between various kinds of machine types and models in the future. This thesis describes how object-oriented programming, design for reusability and common machine control system design patterns can be applied in practice. Even though the system has been designed to be reusable, only the future usage of the new control system will reveal its actual reusability.

ALKUSANAT

Tämä diplomityö on kirjoitettu Bitwise Oy:lle Tampereella, Sandvikille alihankintana tehtävän ohjelmistokehitystyön ohessa. Diplomityön kirjoittamisen lisäksi olen projektin ohessa saanut oppia paljon uutta koneenohjausjärjestelmistä ja niiden kehityksestä.

Haluan kiittää työn ohjaajaa, Juha Tapiolaa, avusta aiheen rajauksessa ja arvokkaasta palautteesta kirjoitusprosessin aikana. Kiitän myös Olli Snellmania, Jaakko Nousiaista ja Ville Lehtistä Sandvikilta työn asiasisällön tarkistamisesta.

Tampereella, 15.8.2018

SISÄLLYSLUETTELO

KUVALUETTELO

TAULUKKOLUETTELO

LYHENTEET JA MERKINNÄT

ACK	engl. Acknowledgement, CAN-datakehyksen kuittausosio.
ADC	engl. Analog to Digital Converter, elektroninen komponentti, joka muuntaa analogisen signaalin digitaaliseen muotoon.
API	engl. Application Programming Interface, ohjelmointirajapinta.
BASE	MCU:n varsinaisen ohjauslogiikan toteuttava applikaatio.
CAN	engl. Controller Area Network, Auto- ja työkoneteollisuudessa yleisesti käytetty väyläprotokolla.
CiA	CAN in Automation, CAN-laitteiden käyttäjien ja valmistajien järjestö.
COB-ID	engl. Communication Object Identifier, CANopen-viestin tunniste.
CODESYS	engl. Controller Development Systems, Smart Software Solutionsin kehittämä IEC 61131-3 -ohjelmointiympäristö.
CPU	engl. Central Processing Unit, tietokoneen tai PLC:n suoritin.
CRC	engl. Cyclic Redundancy Check, CAN-datakehyksen tarkistussumma.
DAC	engl. Digital to Analog Converter, elektroninen komponentti, joka muuntaa digitaalisen luvun analogiseksi signaaliksi.
DC	engl. Direct Current, tasavirta.
DCU	engl. Display Controller Unit, UKKO:n prototyyppikoneen näytönohjausyksikkö.
DM1	engl. Diagnostic Message 1, J1939-protokollan diagnostiikkaviesti.
DTC	engl Diagnostic Trouble Code, DM1:een liittyvä vian yksilöivä koodi.
ECU	engl. Engine Control Unit, moottorinohjausyksikkö.
EMCY	eng. Emergency, CANopen-protokollan viasta ilmoittava protokolla.
Ethernet	ISO/IEC/IEEE 8802-3 -standardin mukainen verkkoteknologia.
FBD	engl. Function Block Diagram, yksi IEC 61131-3 -standardin ohjelmointikielistä.
HAL	engl. Hardware Abstraction Layer, laitteistoabstraktio.
HDS	engl. Hardware Dependent Software, laitteistoriippuvainen ohjelmisto.
HMI	engl. Human-Machine Interface, koneen käyttöliittymä.
IDE-bitti	engl, IDentification Extension, CAN-datakehyksen osa, joka ilmaisee kehyksen arbitraatio-osan pituuden.
IEC	engl. International Electrotechnical Comission, kansainvälinen sähkötekniikan standardisointikomissio.
IL	engl. Instruction List, yksi IEC 61131-3 -standardin ohjelmointikielistä.
IO	engl. Input/Output, ohjausyksikön sisäänmenot ja ulostulot.
ISO	engl. International Standardization Organization, kansainvälinen standardisointiorganisaatio.
LD	engl. Ladder Diagram, yksi IEC 61131-3 -standardin ohjelmointikielistä.
LED	engl. Light Emitting Diode, hohtodiodi.
MC	engl. Machine Control, koneenohjaus.
MCU	engl. Master Controller Unit, UKKO:n prototyyppikoneen keskusohjausyksikkö.
MCUIN	MCU:n Input-applikaatio.

MCUOUT	MCU:n Output-applikaatio.
NMT	engl. Network Management, CANopen-protokollan verkonhallinta-protokolla.
O2M	One To Many, yleisesti käytetty malli ohjausyksiköiden välisen kommunikaation toteuttamiseen.
OSI	engl. Open Systems Interconnection, standardoitu seitsemänkerroksinen malli verkkoprotokollille.
PC	engl. Personal Computer, henkilökohtainen tietokone.
PDO	engl. Process Data Object, CANopen-protokollan viestintäprotokolla.
PDU	engl. Protocol Data Unit, J1939-protokollan datakehys.
PGN	engl. Parameter Group Number, J1939-protokollan parametriryhmän tunniste.
PLC	engl. Programmable Logic Controller, ohjelmoitava logiikkaohjain.
POU	engl. Program Organization Unit, IEC 61131-3 -sovellusten perusrakennusosa.
PWM	engl. Pulse Width Modulation, pulssinleveysmodulaatio.
RAM	engl. Random Access Memory, Tietokoneen tai PLC:n haihtuva muisti.
ROM	engl. Read-Only Memory, Tietokoneen tai PLC:n haihtumaton muisti.
RPDO	engl. Receive PDO, vastaanotettava PDO.
RRC	engl. Radio Remote Controller, radiokauko-ohjain.
RS-232	engl. Recommended Standard 232, eräs sarjaliikenneprotokolla.
RTOS	engl. Real Time Operating System, reaaliaikakäyttöjärjestelmä.
RTR-bitti	engl. Remote Transmission Request, CAN-datakehyksen bitti, joka määrittää datakehyksen tyypin data- tai kyselykehykseksi.
SAE	Society of Automotive Engineers, yhdysvaltalainen autoteollisuuden järjestö.
SDO	engl. Service Data Object, CANopen-protokollan viestintäprotokolla.
SFC	engl. Sequential Function Chart, yksi IEC 61131-3 -standardin ohjelmointikielistä.
SPLC	engl, Safety PLC, turva-PLC.
SPN	engl. Suspect Parameter Number, J1939-protokollan parametrin yksilöllinen tunniste.
SRR-bitti	engl. Substitute Remote Request, osa laajennettua CAN-datakehyksen arbitraatiota.
ST	engl. Structured Text, yksi IEC 61131-3 -standardin ohjelmointikielistä.
SYNC	engl. Synchronization, CANopen-protokollan synkronointiprotokolla.
TCU	engl. Transmission Control Unit, vaihteistonohjausyksikkö.
TP	engl. Transport Protocol, J1939-protokollan usean PDU:n siirtoprotokolla.
TPDO	engl. Transmission PDO, lähetettävä PDO.
TTCAN	engl. Time-Triggered CAN, eräs korkean tason CAN-protokolla.
UKKO	UudelleenKäytettävä KoneenOhjausjärjestelmä, Sandvikin uudesta koneenohjausjärjestelmästä käytettävä nimi tässä diplomityössä.

UML	engl. Unified Modeling Language, Standardoitu mallinnuskieli mm. ohjelmistoarkkitehtuurin mallinnukseen.
USB	engl. Universal Serial Bus, Eräs sarjaliikenneväylätoteutus.
YKOA	Yleinen KoneenOhjausAlusta, Sandvikin konetyyppiriippumattomasta sovelluskehyksestä käytettävä nimi tässä diplomityössä.

1. JOHDANTO

1.1 Diplomityön aihe

Koneet tekevät nykyisin suurimman osan fyysisesti kaikkein raskaimmista töistä. Kehityksen edut ovat ilmeisiä. Koneet tekevät fyysisesti raskaat työt ihmistä paljon nopeammin, edullisemmin ja turvallisemmin. Hyvä esimerkki tästä on kaivosteollisuus. Entisaikaan kaivosmiehet työskentelivät ahtaissa tunneleissa, vaarallisissa työoloissa tehden raskasta työtä hakuin ja ämpärein. Nykyaikaisissa kaivoksissa suuret liikkuvat työkoneet poraavat ja räjäyttävät tilavia tunneleita tai laajoja avolouhoksia, ja kuljettavat louhittua malmia jatkokäsittelyyn kymmenien tonnien kuormissa.

Nykyaikaisessa kaivoksessa ihmisen tehtäväksi jää vain koneiden ohjaaminen, ja sekin työ voidaan tehdä turvallisesti maan pinnalta, tai jopa automatisoidusti. Ensimmäisenä täysin etäohjaukseen siirtynyt maanalainen kaivos oli Northparkesin kaivos Australiassa vuonna 2015 [1]. Koneiden ohjaukseen on kehitteillä automaattisia järjestelmiä, jotka ennen pitkää tekevät ihmisistyöstä lähes tarpeetonta kaivoksilla. Vaikka työn tarve kaivoksissa vähenee, työtä on vielä paljon jäljellä automatisoitujen koneiden kehityksessä.

Kaivostyökoneiden kehityksessä on tärkeää huomioida asiakkaiden erilaiset tarpeet ja maksukyky. Kaivokset ovat keskenään erilaisia, eivätkä kaivosyrittäjät halua maksaa koneiden ominaisuuksista, joita he eivät omalla kaivoksellaan tarvitse. Siksi koneenvalmistajat valmistavat useita eri malleja samasta työkoneesta, ja koneen ominaisuudet voidaan valita yksittäisen asiakkaan tarpeiden mukaan. Tätä varten ei ole järkevää suunnitella ja kehittää jokaista konemallia erikseen alusta alkaen, koska mallit eroistaan huolimatta muistuttavat toisiaan. Vähemmällä kehitystyöllä selvitään, kun konetyyppiä varten kehitetään yksi yleinen koneenohjausjärjestelmä, jota voidaan käyttää aina uudelleen uusia malleja kehitettäessä, ja joka on asetuksin säädettävissä mille tahansa mallille sopivaksi. Muunneltaviksi tarkoitetut ominaisuudet ovat luontevinta toteuttaa ohjelmiston avulla fyysisen laitteiston sijaan. Ohjauslogiikan siirtyminen ohjelmiston vastuulle on ollut jo pitkäaikainen trendi ohjelmiston helpon muokattavuuden vuoksi.

Uudelleenkäytettävän koneenohjausjärjestelmän ohjelmiston suunnittelu ei ole triviaalia. Järjestelmän on huomioitava kaikki mahdolliset eroavaisuudet eri mallien välillä nyt ja tulevaisuudessa, mikä tekee järjestelmästä väistämättä monimutkaisen. Jatkuvasti lisääntyvien vaatimusten toteuttamisen on oltava mahdollista kohtuullisella työllä kohtuullisessa ajassa. Tässä diplomityössä esitellään uudelleenkäytettävän ja helposti laajennetta-

van ohjelmiston yleisiä suunnitteluperiaatteita erityisesti koneenohjausjärjestelmien näkökulmasta. Esiteltäviä menetelmiä ja periaatteita on havainnollistettu käytännön esimerkein.

1.2 Diplomityön rakenne

Diplomityön toinen luku esittelee koneenohjausjärjestelmien, ja etenkin liikkuvien työkoneiden koneenohjausjärjestelmien sovellusalaa. Toisessa luvussa esitellään keskeinen laitteisto koneenohjausjärjestelmän toteuttamiseksi, laitteiston yhteys ulkomaailmaan IO-rajapintansa kautta, laitteiston eri osien väliseen kommunikaatioon käytettävät väylät ja protokollat, sekä lain ja standardien asettamat lisävaatimukset koneenohjausjärjestelmille.

Kolmannessa luvussa esitellään laitteistoläheisen ohjelmoinnin erityispiirteitä ja historiaa. Luvussa esitellään koneenohjausjärjestelmien toteutuksen yleisimmät ohjelmointikielet, joita on käytetty myös luvun 5 esimerkkijärjestelmässä. Luvussa esitellään myös valituista ohjelmointikielistä riippumattomat ohjelmointiparadigmat, joiden johdonmukainen noudattaminen auttaa luomaan helposti ymmärrettävää, jatkokehitettävää ja ylläpidettävää ohjelmistoa.

Neljännessä luvussa perehdytään uudelleenkäytettävän ohjelmiston suunnittelun ja toteutuksen keskeisiin periaatteetteisiin. Luvussa kerrotaan motiivit ohjelmiston uudelleenkäytölle, ja sen merkitys ohjelmistoyrityksen menestykselle. Luvussa kerrotaan, miten yritys voi tehdä ohjelmiston uudelleenkäytöstä systemaattista, maksimoiden uudelleenkäytön hyödyt. Luvussa esitellään myös keskeiset periaatteet uudelleenkäytettävän koneenohjausjärjestelmän itse ohjelmiston suunnitteluun ja toteutukseen käytännönläheisin esimerkein. Neljännen luvun esimerkeillä on pyritty havainnollistamaan esiteltävä periaate mahdollisimman pelkistetyssä ja yksinkertaisessa muodossa, eivätkä ne perustu mihinkään tuntemaani käytännön koneenohjausjärjestelmään.

Viidennessä luvussa esitellään Sandvikin koneenohjausjärjestelmä kaivoslastaus- ja -kuljetuskoneille, jonka kehitystyössä olen ollut mukana alusta alkaen, ja jonka kehitystyön ohessa tämä diplomityö on kirjoitettu. Luvussa kerrotaan, miten toisessa luvussa esitellyt lakien ja standardien turvallisuusvaatimukset toteutuvat, miten järjestelmän osat kommunikoivat keskenään, millä perusteilla kolmannessa luvussa esitellyt ohjelmointikielet ja -paradigmat on valittu, ja miten järjestelmän suunnittelussa vastaan tulleita pulmia on ratkottu soveltamalla neljännessä luvussa esiteltyjä periaatteita. Viidennessä luvussa uudelleenkäytettävän ohjelmiston suunnitteluperiaatteiden noudattamisesta on annettu useita esimerkkejä, jotka perustuvat todelliseen järjestelmään.

Lukujen 4 ja 5 esimerkeissä on käytetty UML-kaavioita (Unified Modeling Language) ohjelmistoarkkitehtuurin esittämiseen. Käytettyjen merkintöjen selitykset on listattu liitteessä A.

2. KONEENOHJAUSJÄRJESTELMÄT

Koneenohjausjärjestelmällä tarkoitetaan laitteistoa ja ohjelmistoa, joka vastaa koneen osien yhteistoiminnasta. Koneenohjausjärjestelmä kuuntelee operaattorin komentoja, ja ohjaa koneen eri toimilaitteita, kuten hydrauliikka- ja jarruventtiilejä, niiden mukaisesti. Koneenohjausjärjestelmä tarkkailee anturien tarjoamaa tietoa koneen tilasta ja ympäristöstä, raportoi vikatilanteet operaattorille, ja tarvittaessa estää koneen vaarallisen käytön. Näiden lisäksi koneenohjausjärjestelmä voi sisältää ominaisuuksia koneen tuottavuuden, käyttömukavuuden, huollon ja diagnostiikan parantamiseksi.

2.1 Koneenohjausjärjestelmän laitteisto

Tässä diplomityössä keskitytään koneenohjausjärjestelmän ohjelmistoon. Lyhyt katsaus koneenohjausjärjestelmän tyypilliseen laitteistoon on kuitenkin tarpeen ohjelmiston erityisvaatimusten ymmärtämiseksi.

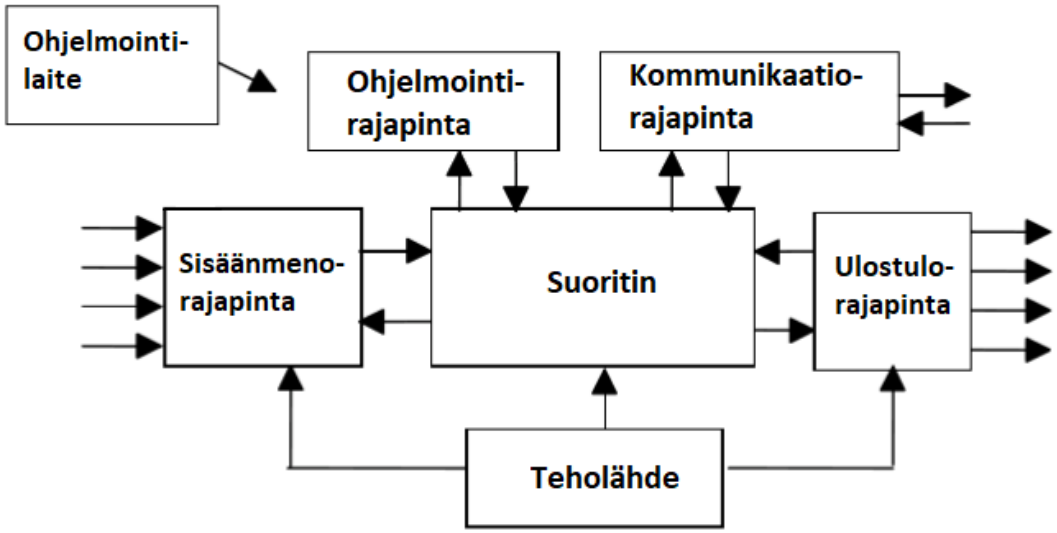
2.1.1 Ohjausyksiköt

Ohjelmiston osuus koko koneenohjausjärjestelmän toteutuksessa on kasvanut aina siitä lähtien, kun ohjelmoitavat logiikkaohjaimet (PLC, Programmable Logic Controller) kehitettiin 1960-luvun lopulla [2][3, s. 1-19]. PLC:llä on paljon yhteistä mikrokontrollerin kanssa, mutta PLC on rakennettu kestämään vaativia ympäristöoloja, ja se sisältää enemmän IO-liitäntöjä (Input/Output, sisäänmenot ja ulostulot) kuin mikrokontrollerit yleensä.

Ennen PLC:n kehitystä koneenohjausjärjestelmät toteutettiin kytkemällä releitä, ajastimia ja muita elektronisia ei-ohjelmoitavia komponentteja yhteen siten, että ne muodostivat halutun ohjauslogiikan. Reikäkorteilla ohjelmoitavat tietokoneet oli keksitty jo 1940-luvulla [4], mutta nämä eivät suuren kokonsa, korkean hintansa ja herkkyytensä vuoksi soveltuneet käytettäväksi useimpien koneiden ohjaamiseen. PLC sen sijaan on pienikokoinen, edullinen ja suunniteltu kestämään tärinää, pölyä, kosteutta, äärimmäisiä lämpötiloja ja sähköistä kohinaa [3, s. 1-19], jotka ovat tyypillisiä työkoneiden käyttöympäristössä.

Ohjelmoitavien komponenttien käytön edut perinteiseen, tuhansilla releillä ja ajastimilla toteutettuun ohjaukseen ovat ilmeisiä ohjauslogiikan kehitystyön helppouden ja uudelleenkäytön kannalta. Muutosten tekeminen ohjelmistoon ja ohjelmiston päivittäminen PLC:lle on vaivatonta ja edullista. Ennen ohjelmoitavia ohjausyksiköitä pienimmänkin muutosten tekeminen ohjauslogiikkaan vaati elektroniikan uudelleensuunnittelun ja -johdotuksen, mikä oli työlästä ja kallista [3, s. 1-19]. PLC voidaan uudelleenohjelmoida täysin uuteen tarkoitukseen, mikä säästää kehitystyön kustannuksia, kun samoja ohjaimia voidaan käyttää uudelleen eri projekteissa.

PLC koostuu sisäisestä teholähteestä, suorittimesta (CPU, Central Processing Unit), ohjelmakoodin ja muun virtakatkoista huolimatta säilyvän tiedon sisältävästä ROM-muistista (Read-Only Memory), ohjelmointirajapinnasta, jonka kautta ohjelmakoodi kirjoitetaan ROM-muistiin, ohjelman ajonaikaisesti käyttämästä RAM-muistista (Random Access Memory) ja IO-rajapinnasta, jonka kautta ulkoiset anturit ja toimilaitteet yhdistyvät PLC:hen. PLC voi sisältää myös muita kommunikaatiorajapintoja, joilla se on yhteydessä muiden PLC:iden ja tietokoneiden kanssa. PLC:n sisäiset osat on esitetty kuvassa 1. [3, s. 1-19]



Kuva 1. *PLC:n sisäinen rakenne [3, s. 1-19].*

Yleiskäyttöisen suorittimensa ansiosta PLC kykenee monimutkaisempaan laskentaan kuin mitä releillä on järkevää toteuttaa. Tämä mahdollistaa hienostuneempien ohjausalgoritmien käytön ja monimutkaisempien ominaisuuksien toteuttamisen materiaalikustannuksia kasvattamatta, kunhan valitun PLC:n suorituskyky ja muistin määrä riittävät. Esimerkki modernista PLC:stä on Epec 3724, jossa on 100 MHz CPU, 768 kilotavua ohjelmamuistia, 1600 kilotavua Flash-muistia muulle pysyvälle tiedolle, ja 1024 kilotavua RAM-muistia [5]. PLC:n suorituskyky ja muistin määrä ovat vaatimattomat PC:hen verrattuna, jossa voi olla esimerkiksi 2 GHz CPU, 8 Gt RAM-muistia ja teratavu pysyvää muistia. Useimpien koneenohjaussovellusten tarpeisiin PLC:n suorituskyky ja muistin määrä kuitenkin riittävät hyvin.

Vaikka PLC:t ovat yleisesti käytössä ohjausyksikköinä autoissa ja työkoneissa, ohjausyksikkönä voi toimia myös reaaliaikakäyttöjärjestelmällä (RTOS, Real-Time Operating System) varustettu, vaativia ympäristöolosuhteita kestämään suunniteltu tietokone, jolla ei välttämättä ole lainkaan omaa IO-rajapintaa. Anturien tilan lukeminen ja toimilaitteiden ohjaaminen tapahtuvat tällöin erillisten IO-moduulien kautta, joihin ohjausyksikkö on yhteydessä esimerkiksi CAN-väylän (Controller Area Network) kautta. IO-moduuli on erityinen ohjausyksikkö, jolla on runsaasti IO-liitäntöjä, oma suoritin IO:n käsittelyyn

ja kommunikaatiorajapinta IO-liitäntöjen tilan kyselyille ja ohjaukselle. IO-moduuli ei sisällä sovelluskohtaista logiikkaa. IO-moduuleja on käytetty ohjausyksikön oman IO-rajapinnan sijaan muun muassa luvun 5 esimerkkijärjestelmässä.

Erillisten IO-moduulien käytön etuna on modulaarisuus. Ohjausyksikkö voi olla esimerkiksi yhden CANopen-protokollaa käyttävän CAN-väylän kautta yhteydessä jopa 126 IO-moduulin kanssa [6]. IO-moduuleissa voi olla yhteensä tuhansia IO-liitäntöjä, joita olisi mahdoton saada mahtumaan yhden PLC:n IO-rajapintaan. IO-moduuleja hyödyntävä ohjausyksikkö skaalautuu modulaarisuutensa ansiosta monen eri kokoisen järjestelmän ohjaamiseen. IO-rajapintoja lukuun ottamatta tietokone sisältää samat osat kuin PLC, joten ne eivät koneenohjausjärjestelmän logiikan suunnittelun ja toteutuksen kannalta eroa toisistaan oleellisesti. Jatkosta molemmista käytetään nimeä ohjausyksikkö.

2.1.2 Ohjausyksikön IO-rajapinta

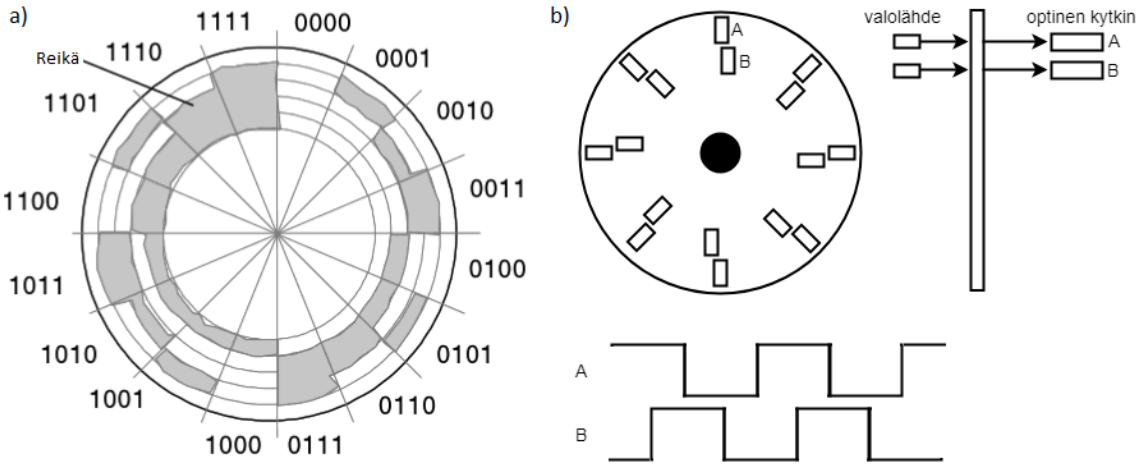
Ohjausyksikön IO-rajapinta tarkoittaa niitä fyysisiä liitäntöjä, joilla ohjausyksikkö kytkeytyy sähköisesti antureihinsa ja toimilaitteisiinsa. Anturit ovat laitteita, jotka muuntavat fysikaalisia ilmiöitä, kuten lämpötilan tai kierrosnopeuden, sähköisiksi signaaleiksi. Vastaavasti toimilaitteet muuttavat sähköisiä signaaleja muiksi fysikaalisiksi ilmiöiksi, kuten valoksi, ääneksi ja liikkeeksi.

Koneenohjausjärjestelmissä tyypillisiä antureita ja ohjaimia ovat muun muassa kytkimet, enkooderit, potentiometrit, paineanturit ja lämpötila-anturit. Anturit voivat tuottaa yhdestä tai useasta bitistä koostuvia diskreettejä eli digitaalisia signaaleja, tai arvoalueeltaan jatkuvia analogisia signaaleja. Signaalin fysikaalinen suure on tyypillisesti jännite, mutta se voi olla myös virta tai taajuus.

Kytkimet tuottavat yhden bitin digitaalisen signaalin. Kytkimiä ovat esimerkiksi operaattorin käyttämät painonapit, sekä asennon tunnistuksessa käytettävät rajakytkimet. Kytkimet ovat usein mekaanisia: kytkimen painaminen sulkee virtapiirin, mikä havaitaan jännitteenä tai virtana ohjausyksikön sisäänmenossa. Toinen yleinen kytkintyyppi on optinen kytkin, jossa valodiodi sallii virran kulun lävitseen, kun sitä valaistaan. Virran kulku katkeaa esineen estäessä valon pääsyn valolähteestä valodiodiin, minkä vuoksi optiset kytkimet soveltuvat hyvin esineiden läsnäolon havaitsemiseen. [3, s. 21-57]

Enkooderit tuottavat useasta bitistä koostuvan digitaalisen signaalin. Enkooderilla voidaan mitata muun muassa absoluuttista kulmaa tai kierrosnopeutta. Kulmaa mittaava absoluuttinen enkooderi koostuu rei'itetystä kiekosta kuvan 2a mukaisesti. Kiekon läpi suunnataan valo, joka osuu vastakkaisella puolella oleviin optisiin kytkimiin. Kiekon asento, ja siten mitattava kulma voidaan tunnistaa rajallisella tarkkuudella kytkinten tilasta. Kuvassa 2b näytetty kulmanopeusanturi perustuu myös rei'itettyyn kiekkoon ja optisiin kytkimiin. Kiekossa on reikiä kahdella eri raidalla. Kummallakin raidalla on sama määrä reikiä, mutta eri raidoilla toisiaan vastaavat reiät ovat hieman eri sektorilla. Kiekon

pyöriessä saadaan kaksi digitaalista signaalia, joiden taajuudesta voidaan päätellä pyörimisnopeus, ja signaalien välisestä vaihe-erosta pyörimissuunta. [3, s. 21-57]



Kuva 2. *Absoluuttisen kulmaenkooderin (a) ja kierrosnopeusenkooderin (b) toimintaperiaate. [3, s. 21-57].*

Potentiometrit, paineanturit ja lämpötila-anturit ovat esimerkkejä analogisista antureista. Potentiometri koostuu säätövastuksesta, jonka resistanssia muutetaan mekaanisella liikkeellä. Resistanssin muutos vaikuttaa anturin jännitehäviöön, josta voidaan päätellä mekaanisen poikkeutuksen suuruus. Käytännön esimerkki potentiometrin käytöstä koneenohjausjärjestelmissä on operaattorin käyttämä ohjaussauva. Paineanturi voi koostua pietsosähköisestä komponentista, joka tuottaa siihen kohdistettuun paineeseen verrannollisen jännitteen. Koneenohjausjärjestelmissä paineantureita tarvitaan esimerkiksi hydrauliikkaöljyn paineen mittaamiseen. Lämpötila-anturi koostuu termistorista, jonka resistanssi ja jännitehäviö muuttuvat lämpötilan mukaan. Lämpötila-anturi voi olla myös lämpölaajenemiseen perustuva kytkin, jolla voidaan havaita ylikuumeneminen, mutta ei mitata lämpötilaa sen tarkemmin. [3, s. 21-57]

Analogiset signaalit on muunnettava digitaaliseen muotoon ennen kuin ohjausyksikön suoritin voi käsitellä niitä. PLC ja IO-moduuli sisältävät yleensä muunnokseen tarvittavan ADC:n (Analog to Digital Converter). ADC muuntaa analogisen signaalin biteillä esitetyksi kokonaisluvuksi, jossa luku 0 vastaa analogisen signaalin arvoalueen minimiarvoa, ja suurin esitettävissä oleva luku (riippuu bittien lukumäärästä) vastaa analogisen signaalin maksimiarvoa. Muunnoksessa menetetään osa alkuperäisen signaalin tarkkuudesta, koska rajallisella määrällä bittejä voidaan esittää vain rajallinen joukko diskreettejä arvoja minimin ja maksimin välillä. Mitä enemmän ADC:ssä on bittejä, sitä tarkempi muunnos on. [3, s. 75-109]

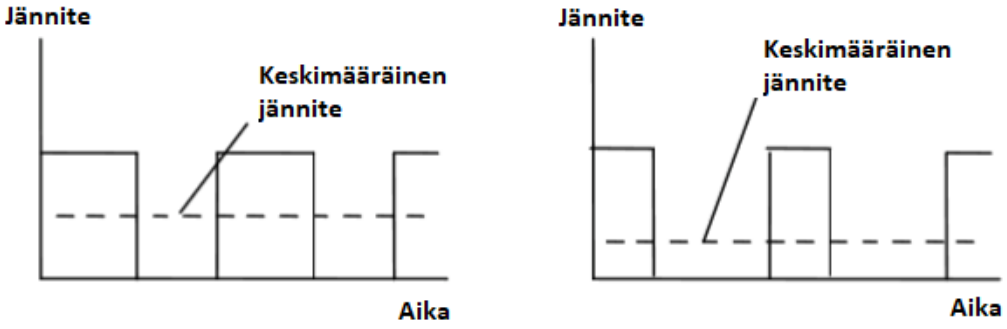
Koneenohjausjärjestelmän tyypillisiä toimilaitteita ovat merkkivalot, releet, solenoidiventtiilit ja DC-moottorit (Direct Current, tasavirta). Toimilaitteet ottavat ohjausyksiköltä komentoja vastaan sähköisinä signaaleina, jotka voivat olla digitaalisia tai analogisia, ja fysikaalisilta suureiltaan jännitettä, virtaa tai taajuutta.

Merkkivalot ja releet ovat esimerkkejä digitaalisilla signaaleilla ohjattavista toimilaitteista. Merkkivalona voi toimia hohtodiodi (LED, Light Emitting Diode), joka on kytketty suoraan ohjausyksikön digitaaliseen ulostuloon. Ohjausyksikkö pystyy yleensä tuottamaan tarpeeksi virtaa ledin tarpeisiin. Esimerkiksi koneen ajovalot sen sijaan tarvitsevat paljon enemmän tehoa, kuin mitä ohjausyksikkö voi antaa. Suuritehoisten digitaalisten toimilaitteiden ohjaamiseen tarvitaan rele. Rele koostuu käämistä, jonka magneettikenttä sulkee sähköisesti erotetun suuritehoisen kytkimen virran kulkiessa käämin läpi. Ohjausyksikön virransyöttökyky riittää kytkimen sulkemiseen tarvittavan käämin magneettikentän tuottamiseksi. [3, s. 21-57]

Solenoidiventtiili on esimerkki analogisella virralla ohjattavasta toimilaitteesta. Solenoidiventtiilejä käytetään koneenohjausjärjestelmissä muun muassa hydrauliikkaöljyn virtauksen ohjaamiseen. Solenoidiventtiili koostuu käämistä, jousesta ja liikkuvasta rautasydämestä. Käämin läpi kulkeva virta saa aikaan magneettikentän, joka aiheuttaa rautasydämeen ohjausvirran suuruuteen verrannollisen mekaanisen voiman. Liikkuessaan rautasydän kaventaa tai laajentaa kanavaa, jota pitkin neste pääsee virtaamaan. Ilman ohjausvirtaa jousi palauttaa rautasydämen alkuperäiseen asentoonsa. [3, s. 21-57]

Ohjausyksikön suoritin pystyy tuottamaan vain digitaalisia signaaleja, joten digitaalinen ohjauskomento on muunnettava analogiseksi virraksi ennen kuin sillä voidaan ohjata solenoidiventtiiliä. PLC ja IO-moduuli sisältävät muunnokseen vaadittavan DAC:n (Digital to Analog Converter). Muunnos on päinvastainen ADC:n sisäänmenosignaaleille tekemälle muunnokselle, ja sen resoluutio on samaan tapaan riippuvainen DAC:n bittien lukumäärästä. [3, s. 75-109]

DC-moottori tuottaa pyörimisliikettä, joka voidaan muuntaa monenlaiseksi muuksi liikkeeksi. DC-moottoria ohjataan yleensä PWM-signaalilla (Pulse Width Modulation, pulssinleveysmodulaatio), joka käytännössä säätää moottorin keskimääräistä tehoa, ja siten tuotetun liikkeen määrää. PWM-signaali on digitaalinen jännitesignaali, ja vaihtelee jaksollisesti loogisen 1- ja 0-tilan (esimerkiksi 5V ja 0V) välillä. Tilojen ajallinen suhde jakson aikana määrää signaalin keskimääräisen tehon kuvan 3 mukaisesti. [3, s. 21-57] PWM on varsin yleinen ohjaussignaalin muoto, ja PLC:t ja IO-moduulit sisältävät tyypillisesti useita PWM-generaattoreita. Esimerkiksi Epec 3724 -PLC sisältää 24 digitaalista ulostuloa, joista kukin voidaan konfiguroida toimimaan PWM-ulostulona [5].



Kuva 3. *PWM-signaali. [3, s. 21-57]*

Ohjausyksikön IO-rajapinta voi siis koostua useista keskenään eri tyyppisistä signaaleista, eikä samaa suuretta mittaava anturi tai samantyyppinen toimilaite välttämättä tuota tai vastaanota samanlaista signaalia. Esimerkiksi lämpötila-antureissa voi olla keskenään huomattavia eroja analogisen signaalin arvoalueen tulkinnassa. Erot erilaisten anturien ja toimilaitteiden ominaisuuksissa on otettava huomioon ohjausjärjestelmän suunnittelussa. Luvussa 4.3 on esitetty ratkaisu IO-rajapinnan yksityiskohtien hallintaan.

2.2 Keskitetty ja hajautettu koneenohjausjärjestelmä

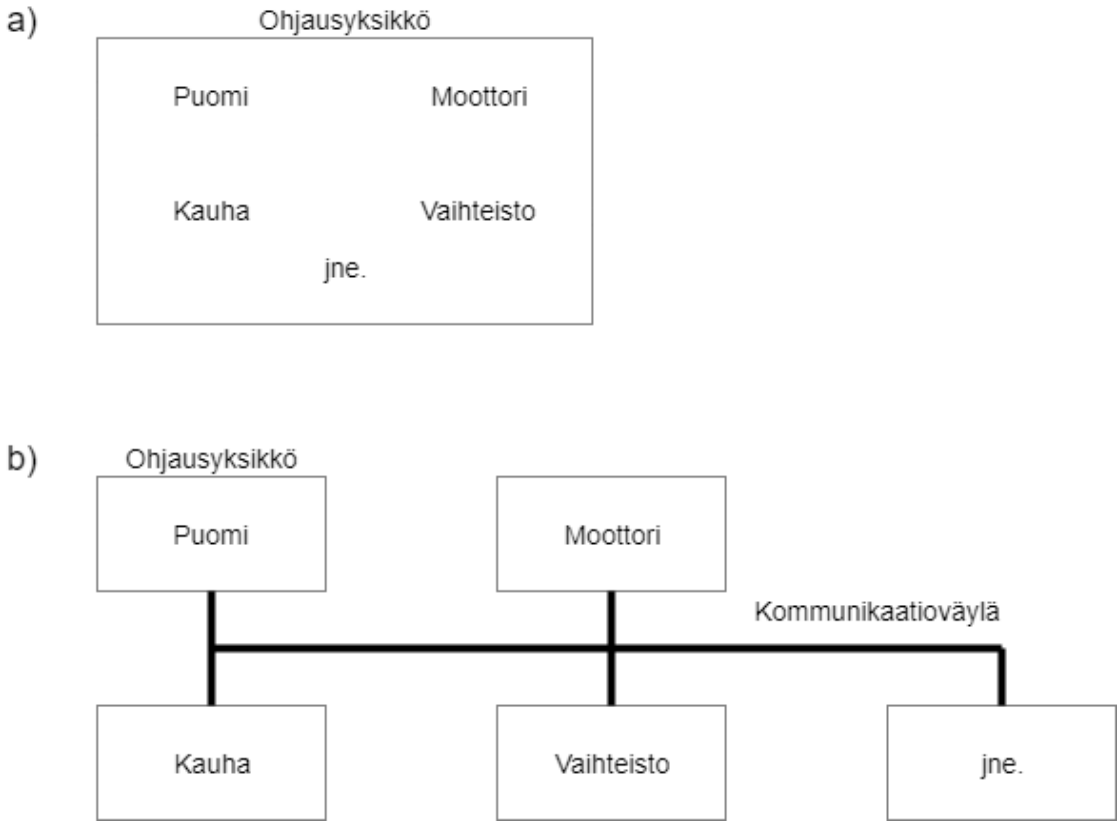
Ohjausyksikön kommunikaatiorajapinnat mahdollistavat viestinnän muiden ohjausyksiköiden ja tietokoneiden kanssa. Ohjausyksikössä voi olla tätä varten esimerkiksi yksi tai useampi CAN-, USB- (Universal Serial Bus), RS-232- (Recommended Standard 232) tai Ethernet-liitäntä. Kommunikaatiorajapinnat mahdollistavat ohjausyksikön helpomman diagnostiikan ja hajautettujen koneenohjausjärjestelmien toteuttamisen.

Koneenohjausjärjestelmää voidaan luonnehtia keskitetyksi tai hajautetuksi sen mukaan, koostuuko ohjausjärjestelmä yhdestä monoliittisesta ohjausyksiköstä, vai useasta keskenään verkotetusta ohjausyksiköstä. Keskitetty koneenohjausjärjestelmä on suunnittelultaan yksinkertainen: Yksi ohjausyksikkö vastaa koko ohjauslogiikasta, ja kommunikoi toimilaitteiden ja anturien kanssa joko suoraan omien IO-liitäntöjensä tai ulkoisten IO-moduulien kautta. Keskitetty koneenohjausjärjestelmä vaatii hajautettua järjestelmää vähemmän kalliita ohjausyksiköitä, eikä suunnittelijan tarvitse huolehtia ohjausyksiköiden välisestä viestinnästä monimutkaisine protokollineen ja lukemattomine virhemahdollisuuksineen. Kuva 4a kuvaa keskitettyä lastauskoneen koneenohjausjärjestelmää. Yksi ohjausyksikkö vastaa kaikkien koneen osien ohjaamisesta.

Koneenohjausjärjestelmän hajauttamiselle on monimutkaisemmissa järjestelmissä useita perusteita. Ensinnäkin koneen suuri koko voi pakottaa jakamaan ohjauslogiikan usealle ohjausyksiköllе siten, että kukin ohjausyksikkö voidaan sijoittaa fyysisesti lähelle niitä toimilaitteita, joita se ohjaa [7, s. 12-31]. Muussa tapauksessa kone vaatisi kohtuuttoman pitkien johdotusten vetämistä toimilaitteiden ja ohjausyksikön välille, mikä hankaloittaisi

kokoonpanoa ja altistaisi sähköisille häiriöille ja johdotusten vikaantumiselle. Toinen syy ohjauslogiikan hajauttamiselle on hajota ja hallitse -periaate: Järjestelmä on helpompi ymmärtää ja toteuttaa, jos se koostuu pienistä, yhden loogisen osakokonaisuuden toteuttavista osista [7, s. 12-31].

Myös halu hyödyntää kolmansien osapuolten tarjoamia valmiita ratkaisuja voi kannustaa tai pakottaa järjestelmän hajauttamiseen. Esimerkiksi kaupalliset dieselmoottorit sisältävät yleensä oman, moottorivalmistajan valmiiksi ohjelmoiman moottorinohjausyksikkönsä (ECU, Engine Control Unit). ECU vastaa vain moottorin sisäisten osien, kuten yksittäisten polttoaine- ja pakokaasuventtiilien ohjaamisesta, ja on riippumaton sovelluksesta, johon moottoria käytetään. Samaa moottoria ja moottorinohjausyksikköä voidaan siten käyttää laajassa kirjossa erilaisia koneita. Vastaavasti sovelluskohtaisen ohjauslogiikan ei tarvitse olla tietoinen moottorin sisäisistä toteutusyksityiskohdista. ECU tarjoaa sovelluskohtaista ohjausta varten kommunikaatiorajapinnan, jonka kautta toinen ohjausyksikkö voi antaa sille yksinkertaisia komentoja, kuten ’käynnistä moottori’, ’aseta moottorin kierrosnopeus’ tai ’kerro moottoriöljyn lämpötila’.



Kuva 4. *Keskitetty (a) ja hajautettu (b) koneenohjausjärjestelmä.*

Kuvan 4a keskitetty lastauskoneen koneenohjausjärjestelmä on esitetty loogisiin osakokonaisuuksiin hajautettuna kuvassa 4b. Kuvan esimerkissä järjestelmän hajautus on viety äärimmilleen: jokaiselle koneen osalle on oma ohjausyksikkönsä. Käytännössä moduulijaon ei tarvitse olla näin hienojakoinen, eikä se ole taloudellisesti järkevää. Järjestelmän suunnittelijan on tehtävä kompromissi modulaarisuuden ja materiaalikustannusten välillä. Kuvan 4b tapauksessa esimerkiksi puomin ja kauhan ohjaus voidaan toteuttaa samalla ohjausyksiköllä, koska niiden ohjaamat toimilaitteet sijaitsevat fyysisesti lähellä toisiaan, ja niillä on loogisesti paljon yhteistä.

2.3 Kommunikaatioväylät ja -protokollat

Hajautetussa koneenohjausjärjestelmässä ohjausyksiköiden toiminta on usein riippuvainen yhdestä tai useammasta muusta ohjausyksiköstä. Esimerkiksi automaattinen vaihteen valinta riippuu moottorin kierrosnopeudesta. Vaihteistonohjausyksikkö (TCU, Transmission Control Unit) tarvitsee siis ECU:lta tiedon kierrosnopeudesta. Ohjausyksiköiden välillä on tätä varten oltava fyysinen kommunikaatioväylä, jota pitkin ne voivat vaihtaa keskenään tietoa sähköisessä muodossa. Lisäksi ohjausyksiköillä on oltava kommunikaatioprotokolla, eli yhteinen käsitys siitä, mitä sähköiset viestit tarkoittavat, kuka saa milloinkin lähettää viestin, miten viestin lähettäminen aloitetaan ja niin edelleen.

2.3.1 Yksi usealle -malli

Kuvan 4b esimerkissä ohjausyksiköiden välille on piirretty yksi kommunikaatioväylä, joka yhdistää kaikkia ohjausyksiköitä. Kaikki ohjausyksiköt eivät välttämättä tarvitse toisiaan, mutta kuvan mukainen järjestely on hyvin yleinen hajautettujen koneenohjausjärjestelmien toteutuksessa. Eloranta, Koskinen, Leppänen ja Reijonen kutsuvat tätä mallia hajautettujen koneenohjausjärjestelmien suunnittelumalleja käsittelevässä kirjassaan Designing Distributed Control Systems - A Pattern Language Approach [7] nimellä Yksi Usealle (One to Many, O2M) [7, s. 131-136]. O2M-mallissa kukin solmupiste, eli väylään yhdistetty laite, voi lähettää väylälle viestejä, ja kaikki muut solmupisteet voivat vastaanottaa ne. Solmupisteen itsensä vastuulle jää seuloa väylällä kulkevista viesteistä sen omaan toimintaan liittyvät viestit.

O2M-mallissa kaikissa lähetetyissä viesteissä on oltava tunniste, jonka perusteella solmupisteet pystyvät erottamaan itseään koskevat viestit muista. Tästä aiheutuu ylimääräistä kuormaa väylälle, ja solmupisteiden on tehtävä ylimääräistä työtä viestien seulomiseksi. Lisäksi O2M-mallissa kaksi solmupistettä voi yrittää lähettää viestejä samanaikaisesti, jolloin ne voivat sekoittua tunnistamattomiksi. O2M-malliin sopivan kommunikaatioprotokollan on kyettävä tunnistamaan ja ratkaisemaan törmäykset viestien lähettämisessä. Kommunikaatioprotokollan viestien formaatin on oltava myös riittävän geneerinen, jotta se soveltuu kaikkien solmupisteiden keskenään hyvin erilaisiin tarpeisiin. Tästä aiheutuu lisää ylimääräistä kuormaa väylälle ja solmupisteille.

Näistä syistä O2M on luonnostaan tehoton verrattuna kahden toisistaan riippuvan solmupisteen välille erikseen tehtävään ad hoc -kytkentään. Ad hoc -kytkennässä törmäysten välttäminen on helppoa, tai törmäyksiin ei tarvitse varautua ollenkaan, jos vain toinen solmupisteistä lähettää tietoa. Ad hoc -kytkennässä lähetettyjen viestien formaatti voidaan optimoida sisältämään vain oleellisen tiedon, jolloin viestien lähettämiseen ja jäsentämiseen kuluu mahdollisimman vähän aikaa.

O2M-mallilla on kuitenkin useita etuja verrattuna ad hoc -kytkentöihin, joiden vuoksi siitä on tullut koneenohjausjärjestelmissä yleinen tapa toteuttaa ohjausyksiköiden, älykkäiden anturien ja muiden solmupisteiden välinen kommunikaatio. Ensinnäkin O2M-malli vaatii vähemmän johdotuksia kuin lukuisat ad hoc -kytkennät, mikä helpottaa koneiden kokoonpanoa. Toinen syy on O2M-mallin joustavuus: uuden solmupisteen lisääminen väylälle ei vaadi lainkaan muutoksia muihin solmupisteisiin. Ad hoc -kytkennöillä uuden solmupisteen lisääminen taas vaatii pahimmillaan vanhojen solmupisteiden korvaamisen uusilla, koska vanhoissa ei ollut enää jäljellä vapaita liitäntöjä uuden ad hoc -kytkennän toteuttamiseksi. Kommunikaation joustavuus on tärkeää, koska koneenvalmistajat haluavat myydä koneisiinsa jälkeenpäin asennettavia lisäosia, ja niiden asennuksen tulisi olla mahdollisimman yksinkertaista.

Yhteinen kommunikaatioväylä, jota pitkin kaikki viestit kulkevat, mahdollistaa myös tehokkaan tiedonkeräyksen. Tietoa voidaan hyödyntää diagnostiikassa, ja tietoa analysoimalla voidaan parantaa koneen käytön taloudellisuutta ja tuottavuutta. Tiedonkeruulaite kuuntelee kaikkia väylän viestejä häiritsemättä muiden väylää käyttävien laitteiden toimintaa millään tavalla. Sandvikin OptiMine [8] on esimerkki CANopen-protokollaa käyttävän CAN-väylän tiedonkeruujärjestelmästä.

2.3.2 CAN-väylä

CAN on O2M-mallin toteuttamiseen soveltuva sarjaliikenneprotokolla. Bosch kehitti CAN-väylän ensimmäisen version 1980-luvun puolivälissä käytettäväksi auton eri osien väliseen kommunikaatioon. Väylän käyttö on sittemmin levinnyt useille eri sovellusaloille, kuten teollisuusautomaatioon, liikkuviin työkoneisiin ja lääketieteellisiin laitteisiin. [37, s. 1-22]

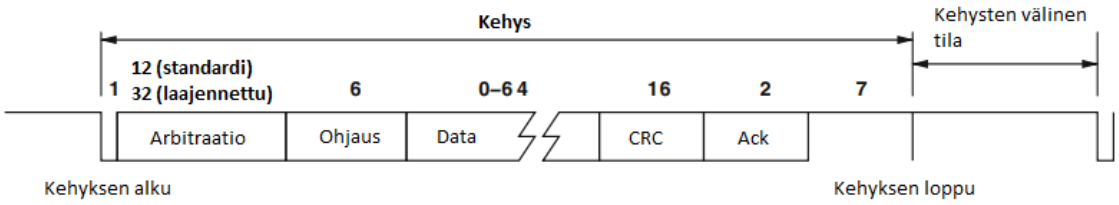
1990-luvun alussa CAN-väylän standardointi siirtyi ISO:n (International Standardization Organization) vastuulle. CAN-väylän standardi on ISO 11898. Standardi koostui enimmillään kuudesta osasta, joiden sisällöt on tiivistetty taulukossa 1. Osat viisi ja kuusi ovat poistuneet standardista vuonna 2017 [34][35]. Standardin osat 1-3 määrittelevät OSI-mallin (Open Systems Interconnection) kaksi alinta kerrosta: siirtokerroksen (data link layer) ja osan fyysisestä kerroksesta. Ensimmäinen osa määrittelee siirtokerroksen ja fyysiseen kerrokseen kuuluvan fyysisen signaloinnin. Toinen ja kolmas osa määrittelevät vaihtoehtoiset toteutukset fyysiselle kerrokselle väylän toteutukseen käytettäviä kaapeleita ja liit-

timiä lukuun ottamatta. Neljäs osa määrittelee korkean tason CAN-protokollan, joka mahdollistaa aikaliipaistun viestinnän CAN-väylällä. Neljännen osan määrittelemä protokolla tunnetaan myös nimellä TTCAN (Time-Triggered CAN) [37, s. 210-215].

Taulukko 1. *ISO 11898 -standardin osat [30][37, s. 1-22]*

Osa	Kuvaus
1 [30]	Standardi CAN-protokollan OSI-mallin siirtokerrokselle ja fyysistä signalointia (physical signaling) koskevalle osalle OSI-mallin fyysisestä kerroksesta.
2 [31]	Standardi CAN-protokollan fyysisen kerroksen nopealle (max. 1Mbit/s) differentiaaliselle parikaapelitoteutukselle.
3 [32]	Standardi CAN-väylän OSI-mallin fyysisen kerroksen hitaalle (max, 125 kbit/s) virhesietokykyiselle toteutukselle.
4 [33]	Standardi CAN-väylän aikaliipaistulle kommunikaatiolle (TTCAN).
5 [34]	Standardi osan 2 fyysisen kerroksen eri tehotasoille. Poistunut standardista 2017.
6 [35]	Standardi osan 2 fyysisen kerroksen valikoivalle herätykselle. Poistunut standardista 2017.

OSI-mallin toiseksi alin siirtokerros määrittelee, missä muodossa tietoa siirretään väylällä, ja millä protokollalla tiedon siirtäminen tapahtuu. ISO 11898-1 määrittelee väylälle lähetettävien kehysten rakenteen, törmäysten välttämisen, viestien priorisoinnin, virhetilanteista ilmoittamisen ja niistä toipumisen. CAN-väylällä erilaisia kehyksiä, eli väylälle yhdellä kertaa lähetettäviä datapaketteja, ovat pääasialliseen viestintään käytettävät datakehykset (data frame), virheistä ilmoittavat virhekehykset (error frame) ja solmupisteen ylikuormittumisesta ilmoittavat ylikuormakehykset (overload frame). Datakehysten rakenne on esitetty kuvassa 5. Datakehys koostuu aloitusbitistä, arbitraatio-osasta, ohjausosasta, itse lähetettävästä datasta, CRC-osasta (Cyclic Redundancy Check), Ack-osasta (ACKnowledgement) ja kehyksen loppuosasta. [37, s. 1-22]



Kuva 5. *CAN-datakehyksen rakenne [37, s. 1-22].*

Datakehyksen arbitraatio-osa on alkuperäisen standardin mukaan 12 bittiä pitkä, ja laajennetun standardin mukaan 32 bittiä pitkä. 12-bittinen arbitraatio-osa koostuu viestin yksilöivästä 11 bitin tunnisteesta, ja viimeinen bitti on RTR-bitti (Remote Transmission Request). RTR-bitti määrittää, onko kyseessä datan lähetys (RTR = 0) vai pyyntö toiselle solmupisteelle lähettää tunnisteen yksilöimä viesti (RTR = 1). Laajennetussa standardissa RTR-bitti on viimeisenä, ja arbitraation 12:s bitti on SRR-bitti (Substitute Remote Re-

quest), joka takaa yhteensopivuuden alkuperäisen ja laajennetun standardin välillä. Laajennetussa arbitraatiossa SRR:n jälkeen on IDE-bitti (IDentifier Extension), joka määrittää, onko kehys alkuperäisen vai laajennetun standardin mukainen (1 = laajennettu). Loput laajennetun standardin tunnisteesta tulevat IDE-bitin jälkeen. Alkuperäisessä standardissa IDE-bitti on ohjausosan alussa, joten molemmissa standardeissa 13 ensimmäistä bittiä ovat semanttisesti samat. [37, s. 1-22]

Arbitraatio-osa määrää viestin prioriteetin. Mikäli usea solmupiste yrittää samanaikaisesti lähettää viestiä, tunnisteeltaan pienemmän viestin lähettäjä saa luvan jatkaa. Pienemmillä tunnisteilla on siis korkeampi prioriteetti. IDE-bitin ansiosta alkuperäisen standardin mukaisilla kehyksillä on korkeampi prioriteetti kuin laajennetun standardin kehyksillä. Korkean tason CAN-protokollissa arbitraatio-osalla voi olla viestien yksilöinnin ja priorisoinnin lisäksi myös muita merkityksiä. [37, s. 1-22]

Datakehyksen ohjausosa on kuusi bittiä pitkä. Alkuperäisessä standardissa ensimmäinen bitti on edellä mainittu IDE-bitti. Toinen bitti on varattu käytettäväksi tulevaisuudessa. Laajennetussa standardissa kaksi ensimmäistä bittiä ovat varattuja. Neljä viimeistä bittiä määrittävät ohjausosaa seuraavan datan pituuden, joka voi olla 0-8 tavua. CRC-osa toimii tarkistussummana, jonka avulla viestin vastaanottavat solmupisteet voivat (todennäköisesti) varmistua siitä, että kehyksessä aiemmin lähetetty data vastaa sitä, mitä lähettäjä halusi lähettää. Mikäli solmupiste vastaanotti viestin, ja se on CRC:n mukaan validi, vastaanottaja kuittaa viestin Ack-osassa. [37, s. 1-22]

ISO 11898-2 ja ISO 11898-3 määrittelevät keskenään vaihtoehtoiset sähköiset vaatimukset CAN-väylän fyysiselle toteutukselle. Standardit eivät kuitenkaan määrittele kaapelien ja liittimien muotoa tai materiaaleja, vaan niille on olemassa omat standardinsa. Standardit määrittelevät väylän topologian, maksimipituuden, sallitut jännitetasot ja signaalien heijastumista ehkäisevien päätevastusten resistanssit. ISO 11898-2 on yleisimmin käytetty standardi CAN-väylän fyysisen kerroksen toteutukselle. ISO 11898 -standardin osien 2 ja 3 lisäksi on olemassa myös SAE J2411 -standardi yksijohtimiselle CAN-väylälle, sekä ISO 11992 -standardi kahden solmupisteen väliselle CAN-väylälle. [37, s. 1-22]

ISO 11899 -standardin mukainen CAN-väylä ei toteuta OSI-mallin ylempiä kerroksia, minkä vuoksi CAN-väylälle on kehitetty useita korkean tason protokollia helpottamaan CAN-väylän käyttöä eri sovellusaloilla. Taulukossa 2 on listattu joitakin yleisimpiä korkean tason CAN-protokollia tyypillisine sovellusaloineen. Tässä diplomityössä kuvataan tarkemmin vain CANopen - ja SAE J1939 -protokollia, koska niitä on käytetty luvun 5 esimerkkijärjestelmässä.

Taulukko 2. *Yleisiä korkean tason CAN-protokollia.*

Protokolla	Ylläpitävä organisaatio	Tyypillisiä sovellusaloja
CANopen	CAN in Automation [38]	Lääketieteelliset laitteet, maastoajoneuvot, merenkulkuelektroniikka, rakennusautomaatio.
SAE J1939	SAE International [39]	Moottorit, pumput, ajoneuvot, satamanosturit
CCP	ASAM [40]	Mittaus, kalibrointi ja diagnostiikka
TTCAN	ISO (ISO 11898-4) [33]	Mahdollistaa deterministisen viestien ajoituksen
ISOBUS	ISO (ISO 11783) [42]	Maa- ja metsätalouden työkoneet
DeviceNet	ODVA [41]	Teollisuusautomaatio

2.3.3 CANopen-protokolla

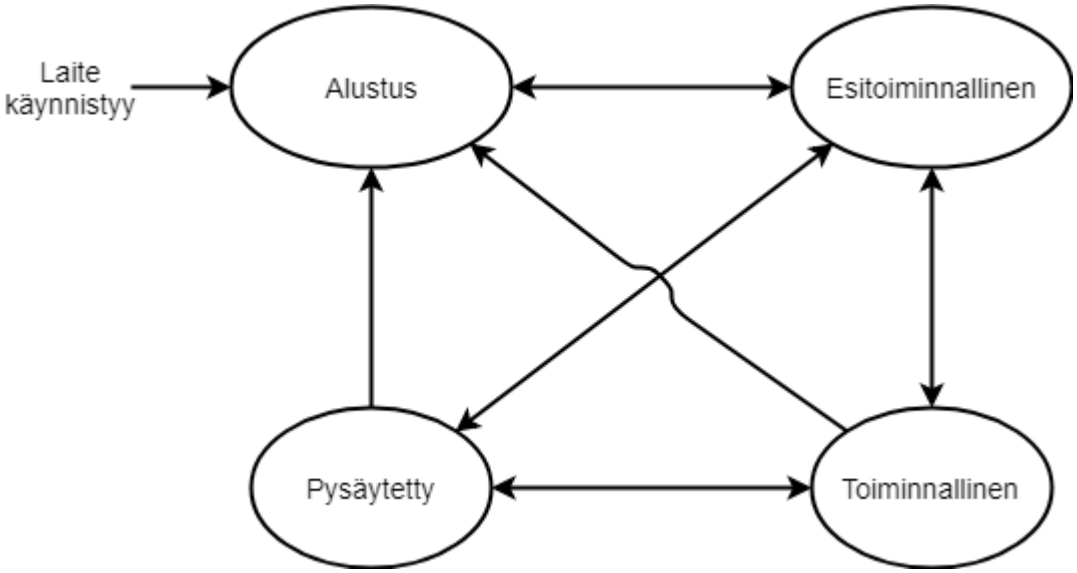
CANopen on CAN in Automation -järjestön (CiA) kehittämä ja ylläpitämä korkean tason protokolla CAN-väylälle. CiA on usean automaatio- ja koneenrakennusalan yrityksen muodostama järjestö, joka perustettiin 1992 kehittämään korkean tason laitevalmistajasta riippumatonta protokollaa CAN-väylälle. CANopen-protokollan ensimmäinen versio julkaistiin vuonna 1994. Protokollan viimeisin versio, CiA 301 CANopen application layer and communication profile 4.2, julkaistiin CiA:n ulkopuolisten saataville vuonna 2011. [9]

CANopen käyttää 12-bittistä datakehyksen arbitraatiota. Arbitraation 11-bittisellä tunnisteella, jota CANopen-protokollan sanastossa kutsutaan COB-ID:ksi (Communication OBject IDentifier), on CANopen-protokollassa viestin yksilöinnin ja priorisoinnin lisäksi kaksi muuta merkitystä. Tunnisteen neljä ylintä (ensimmäisenä lähetettävää) bittiä muodostavat funktiokoodin, joka kertoo viestin protokollan. Funktiokoodit merkityksineen on listattu taulukossa 3. Loput seitsemän bittiä muodostavat viestiin liittyvän solmun tunnisteen, jonka on oltava yksilöllinen saman väylän muihin solmuihin nähden [36]. Bittien järjestyksestä johtuen CANopen-protokollassa viestit priorisoituvat ensisijaisesti funktiokoodin mukaan, ja toissijaisesti solmun tunnisteen mukaan. [37, s. 190-206]

Taulukko 3. *CANopen-protokollan viestityypit [37, s. 190-206]*

Funktiokoodi	Solmun tunniste	COB-ID	Merkitys
0000	0 – 7Fh	0 – 7Fh	NMT-viestit
0001	0	80h	SYNC-viesti
0001	1h – 7Fh	81h – FFh	EMCY-viestit
0010	0	0	Aikaleimaviesti
0011 - 1010	1h – 7Fh	181h – 57Fh	PDO-viestit
1011	1h – 7Fh	581h – 5FFh	SDO-vastaus
1100	1h – 7Fh	601h – 67Fh	SDO-kysely
1110	1h – 7Fh	701h – 77Fh	NMT-virheenkäsittelyn viestit

CANopen-protokollassa on useita sisäisiä protokollia, joiden avulla solmupisteet kommunikoivat keskenään. Eri protokollan viesteillä on eri funktiokoodi, ja siten eri prioriteetti. Kaikkein korkeimmalla prioriteetilla on NMT-protokollan (Network ManagemenT) viestit. CANopen-protokollassa väylällä yksi solmupiste on niin sanottu NMT-master, joka hallinnoi muita väylän solmupisteitä lähettämällä NMT-viestejä. Solmupiste siirtyy käynnistyessään esitoiminnalliseen tilaan (pre-operational), jossa se ei lähetä eikä vastaanota tavanomaisia PDO-viestejään (Process Data Object), mutta sen konfiguraatiota voidaan muunnella SDO-protokollan (Service Data Object) viesteillä. NMT-master voi käskeä solmupisteet esitoiminnallisesta tilasta toiminnalliseen tilaan (operational) yhdessä tai erikseen. NMT-master yksilöi NMT-viestin vastaanottajan COB-ID:n solmun tunnisteessa (0 = kaikki solmut). Toiminnalliseen tilaan siirtyessään solmupiste aloittaa normaalin toimintansa. NMT-master voi käskeä solmupisteitä yhdessä tai yksitellen siirtymään takaisin esitoiminnalliseen tilaan tai pysäyttää ne kokonaan. Pysäytettynä solmupisteet eivät lähetä eivätkä vastaanota viestejä, ennen kuin NMT-master käskee ne takaisin toiminnalliseen tai esitoiminnalliseen tilaan, tai alustamaan itsensä uudelleen. Jokaisen solmupisteen on siis noudatettava kuvan 6 mukaista tilakonetta, ja vaihdettava tilaansa NMT-masterin käskystä. [37, s. 190-206]



Kuva 6. *CANopen-solmujen NMT-tilakone. Perustuu lähteeseen [37, s. 190-206].*

CANopen-protokollassa NMT-master valvoo muita solmupisteitä ja varmistaa, että ne ovat edelleen toiminnassa. Tarkistukseen on kaksi menetelmää. Solmut voivat lähettää itse säännöllisesti sykeviestiä (heartbeat), joka ilmoittaa solmun nykyisen NMT-tilan. Toinen tapa on, että NMT-master kysyy säännöllisesti solmuilta niiden NMT-tilan. [43]

Toisena protokollien prioriteettijärjestyksessä on SYNC-protokolla (SYNChronization), joka mahdollistaa solmupisteiden synkronisoidun toiminnan (esim. PDO-viestien lähettämisen) synkronointia hallinnoivan solmun (voi olla eri kuin NMT-master) lähettäessä

synkronointiviestin. Kolmantena prioriteettijärjestyksessä ovat EMCY-viestit (EMergenCY), joilla solmupisteet ilmoittavat sisäisistä virheistään muille solmuille. Neljäntenä prioriteettijärjestyksessä on aikaleimaviesti (timestamp), jolla väylän solmujen kellot synkronoidaan. Yksi solmuista (voi olla eri kuin NMT-master) lähettää aikaleimaviestin, joka sisältää yhteisen kellonajan ilmoitettuna millisekunteina referenssiajasta (keskiyö 1.1.1984). [37, s. 190-206]

Kullakin väylän solmulla on niin kutsuttu objektikirjasto, joka määrittelee solmun CANopen-rajapinnan. Objektikirjasto määrittelee kaikki solmun dataobjektit eli tiedot, joita muut solmut voivat pyytää luettavaksi tai kirjoitettavaksi. Osa objektikirjaston sisällöstä on standardoitu tai varattu tulevaan käyttöön, mutta laitevalmistajalla on objektikirjastossa runsaasti tilaa laitekohtaisille objekteille. Standardoituja objekteja ovat muun muassa tietotyyppien määrittelyt, PDO-sidonnat, laitevalmistajan ja laitteen tyypin yksilöivät tunnisteet ja solmutunniste väylällä. Objekteilla on laitteen objektikirjastossa 16-bittinen yksilöllinen indeksi, sekä mahdollisesti alaindeksi väliltä 1 - 254, joiden perusteella muut solmupisteet voivat lukea tai kirjoittaa objekteja. Väylän muut solmut voivat lukea ja kirjoittaa solmun objektikirjaston objekteja SDO- ja PDO-protokollilla. [37, s. 190-206]

SDO-protokolla mahdollistaa objektien lukemisen ja kirjoittamisen minä tahansa hetkenä, kunhan solmu ei ole pysäytetyssä tilassa. Solmu ei koskaan lähetä oma-aloitteisesti omiin objekteihinsa liittyviä SDO-viestejä, vaan SDO-protokollassa NMT-master pyytää objektin lukua tai kirjoitusta lähettämällä väylälle SDO-kyselyn, jonka tunnisteessa on määritelty kyselyn kohteena oleva solmu, ja jonka datakentässä on määritelty kyselyn tyyppi (luku/kirjoitus), kyselyn kohteena olevan objektin indeksi ja alaindeksi, sekä mahdollisesti objektiin kirjoitettava data. Kyselyn kohteena oleva solmu lähettää vuorostaan SDO-vastauksen, jonka tunnisteessa on solmun oma tunniste, ja jonka datassa on kyselyn kohteena olevan objektin indeksi ja alaindeksi, tieto kyselyn tilasta, ja mahdollisesti kysyjälle lähetettävä objektin sisältämä data. SDO-protokollan luku- ja kirjoitusoperaatio voi vaatia useita peräkkäisiä viestejä kyselyn kohteena olevan objektin datan koosta riippuen [37, s. 190-206]. SDO-protokollaa ei suositella käytettäväksi järjestelmän tavanomaiseen viestintään, koska runsaat ja satunnaiset SDO-kyselyt tekevät väylän kuorman ennakoimisesta ja mitoittamisesta vaikeaa.

SDO-kyselyjen sijaan järjestelmän tavanomaiseen viestintään kannattaa käyttää PDO-protokollaa. PDO-protokollassa kullakin solmulla on RPDO-objekteja (Receive PDO), joiden avulla muut solmut voivat kirjoittaa solmun objektikirjaston objekteja, ja TPDO-objekteja (Transmit PDO), joita solmu lähettää väylälle. Kukin PDO voi sisältää enimmillään kahdeksan tavua dataa. Kuhunkin PDO-objektiin on sidottu objektien koosta riippuen 1-64 objektikirjaston objektia. Solmu voi lähettää TPDO-viestejä väylälle, jos jokin TPDO-viestiin sidottu objekti muuttuu tai edellisen TPDO-viestin lähettämisestä on kulunut tietty aika. Solmu voi vastaanottaa RPDO-objektiin kohdistuvia kirjoituspyyntöjä koska tahansa. [37, s. 190-206]. Etuna PDO-protokollassa SDO-protokollaan nähden on

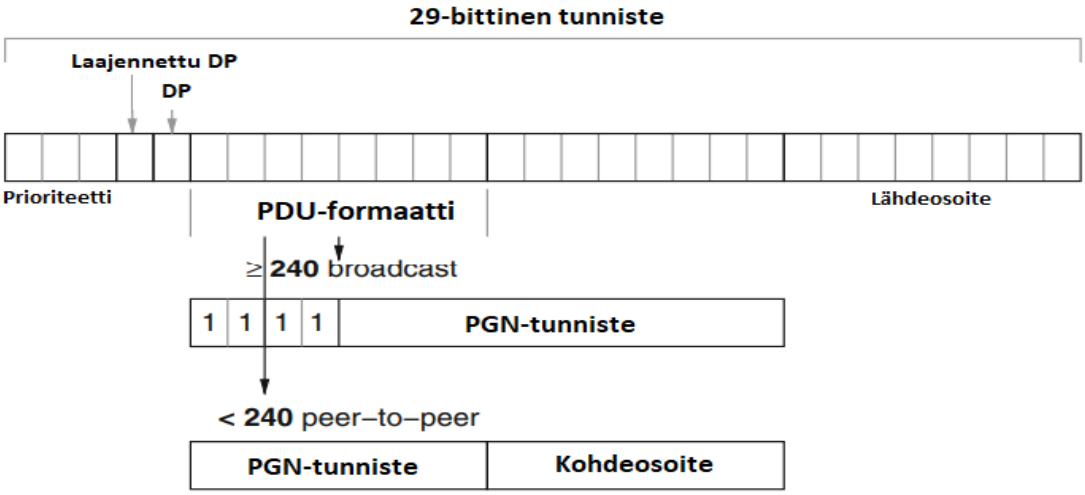
pienempi ja ennakoitavampi väyläkuorma. Tapahtumapohjaisen TPDO-viestien lähetyksen avulla vältetään objektien turhaa pollaamista, mikä SDO-protokollassa olisi välttämätöntä. Käyttämällä tietyn ajan välein toistuvaa TPDO-viestien lähetystä väylän kuormitus voidaan suunnitella etukäteen.

2.3.4 SAE J1939 -protokolla

SAE J1939 on korkean tason CAN-protokolla, jonka kehittäjä ja ylläpitäjä on SAE International (Society of Automotive Engineers). J1939 kehitettiin alun perin raskaiden ajoneuvojen kommunikaatioon, mutta protokollan käyttö on laajentunut useille muille sovellusaloille. J1939-standardi koostuu useasta osasta. Tällä hetkellä standardin osia on viisi: J1939-1x (fyysinen kerros), J1939-2x (siirtokerros), J1939-3x (verkkokerros), J1939-7x (sovelluskerros) ja J1939-8x (verkonhallinta). Puuttuvat osat 4 – 6 on varattu määriteltäväksi myöhemmin, ja ne määrittelevät OSI-mallin kuljetuskerroksen (transportation layer), yhteysjaksokerroksen (session layer) ja esitystapakerroksen (presentation layer). [37, s. 181-190]

J1939-standardin fyysinen kerros perustuu ISO 11898-2 -standardiin, mutta asettaa sille rajoitteita: J1939:n ainoa sallittu baudinopeus on 250 kbit/s, kaapelin maksimipituus on 40 m, väylä ei saa haaroittua, ja suurin sallittu solmujen lukumäärä väylällä on 30. J1939:n fyysinen kerros määrittelee myös kaapelien ja liittimien ominaisuudet, joita ISO 11898 -standardi ei määrittele. [37, s. 181-190]

J1939:n siirtokerros käyttää laajennettua versiota ISO 11898-1 -standardista, eli 32-bittistä datakehyksen arbitraatio-osaa. Toistaiseksi J1939 on ainoa korkean tason CAN-protokolla, joka hyödyntää laajennettua standardia, joka alun perin toteutettiin juuri SAE:n pyynnöstä. J1939-protokollassa datakehystä kutsutaan PDU:ksi (Protocol Data Unit). PDU:n arbitraatio-osa koostuu 29 bitin tunnisteesta ja luvussa 2.3.2 mainituista RTR-SRR- ja IDE-biteistä. Tunniste koostuu kolmen bitin prioriteetista, kahdesta DP-bitistä (Data Page), 16 bitin parametriryhmänumerosta (PGN, Parameter Group Number) ja 8 bitin lähdeosoitteesta. Tunnisteen rakenne on esitetty kuvassa 7. [37, s. 181-190][44]



Kuva 7. *J1939-datakehyksen tunniste. Perustuu lähteisiin [37, s. 181-190] ja [44].*

Mitä pienempi tunnisteen prioriteettikentän arvo on, sitä korkeampi prioriteetti viestillä on. DP-bitit mahdollistavat PGN-tunnisteiden uudelleenkäytön myöhemmissä standardin laajennoksissa. 16-bittinen PGN määrittelee PDU:n tyypin, PGN-tunnisteen ja mahdollisen kohdeosoitteen. PGN:n neljä ylintä bittiä määrittävät, onko kyseessä kaikille väylän solmuille lähetettävä (broadcast) viesti, vai ainoastaan tietylle solmulle lähetettävä (peer-to-peer) viesti. Broadcast-viestissä PGN:n neljä ylintä bittiä ovat arvoltaan 1, ja loput 12 bittiä määrittävät PGN-tunnisteen. Peer-to-peer -viestissä PGN:n ensimmäinen tavu määrittää PGN-tunnisteen ja toinen tavu viestin kohteena olevan solmun osoitteen. Peer-to-peer -viesteille on siten käytettävissä 240, ja broadcastviesteille 4096 PGN-tunnistetta jokaista DP-bittien variaatiota kohden. Tunnisteen alin tavu sisältää PDU:n lähettäneen solmun osoitteen. [37, s. 181-190][44]

PGN yksilöi parametriryhmän, joka koostuu useista toisiinsa liittyvistä parametreista, jotka lähetetään väylälle aina samassa PDU:ssa. Parametrien arvot muodostavat PDU:n datakentän. PDU:n datakentän tulkinta on tehtävä parametriryhmän määrittelyn mukaan. Parametriryhmän määrittely voi olla joko osa J1939-standardia tai valmistajakohtainen. J1939 määrittelee raskaiden ajoneuvojen sovellusalalla yleisesti käytettyjä parametriryhmiä, mikä takaa yhteensopivuuden eri valmistajien vastaavien tuotteiden välillä. Yhteensopivuuden ansiosta esimerkiksi kaivoslastauskoneen valmistaja voi helposti vaihtaa dieselmoottorin toisen valmistajan moottoriin, jos molempien moottorien ohjausyksiköt noudattavat J1939-standardia CAN-rajapintansa toteutuksessa. J1939-standardissa laitevalmistajille on varattu PGN-tunnisteita, joita ne voivat käyttää omille täydentäville parametriryhmilleen. Kuva 8 näyttää esimerkin yhden J1939-standardin parametriryhmän määrittelystä. [37, s. 181-190][44]

Name:		Engine temperature 1 – ET1
Transmission rate:		1s
Data length:		8 bytes
Extended Data Page		0
Data page:		0
PDU format:		254
PDU specific:		238
Default priority:		6
PG Number:		65,262 ($00FEEE_{16}$)
Description of data:		
Byte:	1	Engine Coolant Temperature
	2	Engine Fuel Temperature 1
	3,4	Engine Oil Temperature 1
	5,6	Engine Turbocharger Oil Temperature
	7	Engine Intercooler Temperature
	8	Engine Intercooler Thermostat Opening

Kuva 8. *Esimerkki J1939-standardin parametriryhmän määrittelystä [44].*

Jokaiselle parametriryhmän parametrille on määritelty J1939-standardissa tai laitteen datalehdellä yksilöllinen SPN-tunniste (Suspect Parameter Number). Standardi tai datalehti ilmoittavat kunkin SPN:n yksilöimän parametrin muunnoskaavat, joiden mukaan parametri on muunnettava PDU:n sisältämästä raaka-arvosta sen todellisiksi yksiköiksi ja päinvastoin. SPN-tunnistetta käytetään myös J1939-protokollan diagnostiikassa. Jos J1939-solmu tukee diagnostiikkaa, jokaisen solmun J1939-sovelluksen (voi olla useampi kuin yksi) on säännöllisesti lähetettävä standardissa määritetty DM1-parametriryhmän (Diagnostic Message 1) viesti, jolla se ilmoittaa nykyisestä tilastaan. Sovellukset ilmoittavat parametreihin liittyvistä vioista tai hälyttävistä arvoista lähettämällä DM1-viestin mukana listan DTC-koodeja (Diagnostic Trouble Code), jotka määrittelevät vikaan liittyvän SPN:n, sekä vian tyypin ja vakavuuden. [44]

Jos vikoja on useampi kuin yksi, DTC-lista ei mahdu yhteen DM1-viestiin, vaan lähettäjän on käytettävä J1939-standardin verkkokerroksessa määriteltyä siirtoprotokollaa (TP, Transport Protocol) usean yhteenkuuluvan viestin lähettämiseksi [44]. TP:tä on mahdollista käyttää myös muiden kuin diagnostiikkaviestien lähettämiseen, ja se tukee jopa 1785 tavun lähettämistä yhdessä sarjassa viestejä [37, s. 181-190].

J1939-protokollasssa jokaisella väylän solmulla on 8-bittinen osoite, jota käytetään PDU-viestien lähde- ja kohdeosoitteina. J1939 verkonhallintaprotokolla mahdollistaa osoitteiden dynaamisen jakamisen [37, s. 181-190]. Yleensä järjestelmän suunnittelija kuitenkin asettaa kullekin solmulle staattisen osoitteen, joten verkonhallintaprotokollan dynaamista osoitteiden jakamista ei käsitellä tässä tarkemmin.

2.4 Lain ja standardien vaikutus koneenohjausjärjestelmään

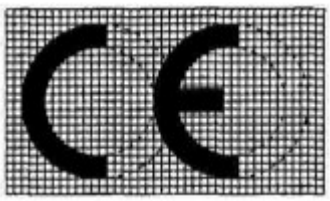
Tässä aliluvussa käsitellään lainsäädännön asettamia vaatimuksia koneenohjausjärjestelmien ominaisuuksille, suunnittelulle, toteutukselle, varmentamiselle ja dokumentoinnille.

Lisäksi tarkastellaan kaivostyökoneisiin liittyviä teollisuusalakohtaisia vapaaehtoisia turvallisuusstandardeja.

2.4.1 EU:n konedirektiivi

Työkoneet voivat vikaantuessaan tai väärin tai huolimattomasti käytettyinä aiheuttaa vakavaa vaaraa ihmisten terveydelle tai hengelle. Koneiden turvallisuuteen on tästä syystä kiinnitetty erityistä huomiota lainsäädännössä. EU-alueella tärkein koneiden turvallisuutta koskeva säädös on EU:n konedirektiivi, eli Euroopan parlamentin ja neuvoston direktiivi 2006/42/EY [10]. Konedirektiivi koskee 'koneita', joilla tarkoitetaan "toisiinsa liitettyjen osien tai komponenttien yhdistelmää, jossa on tai joka on tarkoitettu varustettavaksi muulla kuin välittömällä ihmis- tai eläinvoimalla toimivalla voimansiirtojärjestelmällä ja jossa ainakin yksi osa tai komponentti on liikkuva ja joka on kokoonpantu erityistä toimintoa varten" [10]. Direktiivi määrittelee koneiden turvallisuusvaatimukset, jotka kaikkien EU-alueella myytävien ja käytettävien koneiden on toteutettava. Se määrittelee myös valmistajien ja paikallisten viranomaisten vastuut direktiivin vaatimusten toteutuksessa ja valvonnassa. Suomessa EU:n konedirektiivin toteuttaa valtioneuvoston asetus koneiden turvallisuudesta 12.6.2008/400 [11].

Direktiivin vaatimusten toteuttaminen ja säädöstenmukaisuuden vakuuttaminen ovat koneenvalmistajan vastuulla. Valmistaja vakuuttaa koneen täyttävän konedirektiivin vaatimukset kiinnittämällä siihen näkyville ja pysyvästi kuvan 9 mukaisen CE-merkinnän. Viranomaisella ei ole oikeutta vaatia CE-merkinnän lisäksi muuta todistusta koneen turvallisuudesta sen myynnin ja käytön sallimiseksi. Konedirektiivi sisältää vaatimuksenmukaisuusolettaman, jonka mukaan viranomaisen on oletettava koneen täyttävän turvallisuusvaatimukset, jos valmistaja on sen CE-merkinnällä ilmoittanut. Valmistaja on kuitenkin viranomaisen vaatiessa velvollinen esittämään CE-merkintään oikeuttavan teknisen rakennedokumentin. Paikallisen viranomaisen velvollisuus on valvoa, että direktiivin vaatimusten vastaisiksi todetut tuotteet vedetään pois markkinoilta tai niihin tehdään tarvittavat muutokset mahdollisimman nopeasti. Valmistajaa rangaistaan säädöstenmukaisuuden rikkomisesta konedirektiivin toteuttavan paikallisen lainsäädännön mukaisesti. Direktiivin mukaan rangaistuksen on oltava tehokas, oikeasuhteinen ja varoittava. [10]



Kuva 9. *EU:n konedirektiivin vaatima CE-merkintä [10].*

Konedirektiivi asettaa vaatimuksia sekä koneen laitteiston että ohjelmiston suunnittelulle ja toteutukselle. Direktiivi vaatii suunnittelemaan ja rakentamaan ohjausjärjestelmän estämään vaaratilanteiden syntymisen koneen tarkoitetussa käytössä ja käyttöympäristössä.

Vikaantuessaankaan kone ei saa aiheuttaa vaaraa ihmisille. Tämä koskee niin koneen mekaanista vikaantumista, sähkönsyötön häiriöitä, ohjelmistovikoja kuin virheitä ohjauslogiikassa. Koneenohjausjärjestelmän suunnittelussa ja toteutuksessa on kohtuudella varauduttava myös käyttäjän virheisiin. [10]

Kone ei saa käynnistyä odottamattomasti, ja käynnistyksen aiheuttavat ohjaimet on toteutettava siten, että niiden aktivoiminen vaatii käyttäjältä erityistä tarkoituksellisuutta [10]. Tarkoituksellisuuden varmistamiseksi koneenohjausjärjestelmä voi esimerkiksi vaatia moottorin käynnistävää painonappia pidettävän painettuna pidemmän aikaa ennen kuin moottori käynnistetään. Jos koneessa on useampi vaihtoehtoinen työasema, kuten radio-ohjain, jolta kone voidaan käynnistää, koneessa on oltava lisälaite, joka estää useaa käyttäjää aiheuttamasta vaaraa toisilleen [10]. Tällainen lisälaite voi olla esimerkiksi summeri, joka soi tietyn aikaa ennen koneen käynnistymistä, antaen muille käyttäjille aikaa siirtyä pois vaara-alueelta tai estää koneen käynnistyminen.

Koneen on oltava kaikissa tilanteissa pysäytettävissä joko normaaleilla sammutuskomennolla, hätäpysäytysohjaimilla tai automaattisilla toiminnoilla. Pysäytyskäskyn saatuaan koneenohjausjärjestelmä ei saa estää pysäyttämistä riippumatta pysäytyskäskyn lähteestä tai samaan aikaan annetuista ristiriitaisista komennoista. Aktiivinen pysäytyskomento keskeyttää ja estää koneen käynnistymisen. Jos koneessa on useampi vaihtoehtoinen työasema, kone pitää olla mahdollista sammuttaa miltä tahansa niistä. [10]

Hätäpysäytystä varten koneessa on oltava yksi tai useampi selkeästi merkitty ja nopeasti käytettävissä oleva hätäpysäytysohjain. Hätäpysäytyksen tapauksessa koneen uudelleenkäynnistäminen täytyy estää, kunnes hätäpysäytys on kuitattu jollakin erityisellä toimenpiteellä. [10] Vaadittava erityistoimenpide voi olla esimerkiksi aktivoidun hätäpysäytysohjaimen vapauttaminen avaimella, jota säilytetään erillään hätäpysäytysohjaimesta. Toisin kuin tavallisen pysäytyksen, hätäpysäytyksen on pysäytettävä kaikki koneen liikkeet mahdollisimman nopeasti, aiheuttamatta kuitenkaan uusia vaaratilanteita. Hätäpysäytyksen kuittaus ei saa käynnistää konetta automaattisesti uudelleen. [10] Tyypillinen toteutus hätäpysäytysohjaimelle on iso punainen painike, joka katkaisee kaikkien toimilaitteiden sähkönsyötön.

Koneen ohjainten on oltava ergonomisia, selkeästi merkittyjä ja turvalliset käyttää. Koneen turvallisen käytön kannalta välttämättömien mittarien ja muiden osoitinlaitteiden on oltava luettavissa ohjauspaikalta. Ohjauslaitteiden liikkeen on vastattava niiden aikaansaamaa vaikutusta. [10] Viimeinen vaatimus tarkoittaa käytännössä sitä, että ohjaimen liikkeen suuruuden tulee olla verrannollinen sen ohjaaman laitteen liikkeen suuruuteen ilman häiritsevää viivettä. Esimerkiksi kaasupolkimen painallus kasvattaa moottorin kierrosnopeutta välittömästi painalluksen syvyyttä vastaavassa suhteessa.

Direktiivi ottaa vain vähän kantaa koneen toteutusyksityiskohtiin. Valmistajalla on vapaus valita parhaaksi katsomansa toteutustavat, kunhan ne täyttävät direktiivin vaatimukset. Käytännössä jotkin direktiivin vaatimukset pakottavat kuitenkin tekemään tiettyjä suunnittelu- ja toteutusratkaisuja. Esimerkki vaatimus mahdollisuudesta koneen pysäyttämiseen milloin tahansa jättää pysäytysohjaimelle vain kaksi mahdollista toteutustapaa. Ensimmäinen tapa on toteuttaa pysäytysohjain kytkimenä, joka välittömästi katkaisee koneen sähkönsyötön. Mikäli koneen mekaanisten osien suojaamiseksi pysäytys halutaan tehdä hallitummin ohjelmiston avulla, ohjausyksiköiden täytyy pystyä joka tilanteessa reagoimaan pysäytyskomentoon hyvin lyhyellä vasteajalla. Tämä onnistuu vain, jos ohjausyksiöissä on rajalliset vasteajat takaava reaaliaikakäyttöjärjestelmä. Myös direktiivin vaatimus ohjainten viiveettömyydestä ohjaa varustamaan ohjausyksiköt reaaliaikakäyttöjärjestelmällä. Käytännössä koneenohjauksesta vastaavissa ohjausyksiköissä on aina reaaliaikakäyttöjärjestelmä, vaikka direktiivi ei sitä suoraan vaadi.

2.4.2 Kaivostyökoneiden turvallisuusstandardit

EU:n konedirektiivi on yleinen turvallisuusstandardi kaikille direktiivin määrittelemille koneille, jota konevalmistajien on noudatettava sovellusalueesta riippumatta. Yleispätevyytensä vuoksi direktiivi ei ota yksityiskohtaisesti kantaa eri sovellusalojen erityisvaatimuksiin. Monilla sovellusaloilla on siksi lisäksi vapaaehtoisia turvallisuusstandardeja, jotka määrittelevät yksityiskohtaisemmin, miten koneiden turvallisuus on toteutettava. EU-alueella vapaaehtoiset sovellusalakohtaiset standardit täydentävät EU:n konedirektiiviä, mutta eivät aseta laillisesti sitovia lisävaatimuksia.

Vapaaehtoiset turvallisuusstandardit yhtenäistävät ja helpottavat laillisesti sitovien säädösten noudattamista sovellusalalla, ja tarjoavat valmiita ratkaisumalleja laillisesti sitovien vaatimusten toteuttamiseksi. Vapaaehtoisten turvallisuusstandardien noudattaminen on myös kilpailuvaltti, ja parantaa koneenvalmistajan oikeusturvaa. Onnettomuuksien sattuessa valmistaja voi vedota noudattamiinsa alan standardeihin, jotka määrittelevät nykytiedon mukaan parhaat menetelmät onnettomuuksien välttämiseksi. Näistä syistä koneenvalmistajat usein päättävät noudattaa vapaaehtoisia turvallisuusstandardeja.

Luvussa 5 esitelty Sandvikin uusi koneenohjausjärjestelmä ja siihen perustuvat koneet noudattavat useaa vapaaehtoista turvallisuusstandardia. Tärkein niistä on EN 1889-1:2011 -konetyyppistandardi maanalaisille kumipyöräisille kaivostyökoneille [45]. Konetyyppistandardi kokoaa yhteen lukuisissa muissa standardissa määriteltyjä vaatimuksia. Keskeisimmät viitatut standardit sisältöineen on listattu taulukossa 4.

Taulukko 4. *EN 1889-1:2011 -standardin viittaamat muut standardit.*

Standardi	Sisältö
EN ISO 4413:2010 [46]	Hydraulisen voimantuottojärjestelmän ja sen osien yleiset säännöt ja turvallisuusvaatimukset.
EN 60204-1:2006 [47]	Koneen sähköisten varusteiden yleiset vaatimukset.

EN 60529:1992 [48]	Koteloinnin suojaustasot.
EN ISO 3411:2007 [49]	Maansiirtokoneet – Operaattorien fyysiset mitat ja ohjaustilan vähimmäissuojaus.
EN ISO 3449:2005 [50]	Putoavilta esineiltä suojaavat rakenteet – laboratoriotestit ja niiden läpäisyvaatimukset
EN ISO 3450:2008 [51]	Maansiirtokoneet – kumipyöräisten koneiden jarrutusjärjestelmien suorituskykyvaatimukset ja testausmenetelmät.
EN ISO 3471:2008 [52]	Maansiirtokoneet – koneen kaatumiselta suojaavien rakenteiden laboratoriotestit ja niiden läpäisyvaatimukset.
EN ISO 6682:1986 [53]	Maansiirtokoneet – mukavuusalueet ja ohjainten ulottuvilla oleminen.
EN ISO 12100:2010 [54]	Yleiset suunnitteluperiaatteet – riskien arviointi ja pienentäminen.
EN ISO 13849-1:2006 [55]	Ohjausjärjestelmän turvallisuuskriittisten osien yleiset periaatteet.
ISO 5010:2007 [56]	Maansiirtokoneet – kumipyöräisten koneiden ajosuunnan ohjauksen vaatimukset.

EN-1889-1:2011 -konetyyppistandardin lisäksi Sandvikin lastaus- ja kuljetuskoneet noudattavat muun muassa taulukossa 5 listattuja standardeja.

Taulukko 5. *Muita Sandvikin soveltamia turvallisuusstandardeja.*

Standardi	**Sisältö**
EN ISO 13850 [57]	Koneen hätäpysäytys.
EN 62061:2005 [58]	Automaation turvajärjestelmä
EN 60947-1:2007 [59]	Matalajännitteistein sähköisten ohjainten turvallisuusvaatimukset.
AS 4024 [60]	Australiaan vietävien koneiden laillisesti sitovat vaatimukset.
CAN/CSA-M424.2 [61]	Kanadaan vietävien koneiden laillisesti sitovat vaatimukset.

3. LAITTEISTOLÄHEINEN OHJELMISTO

Ohjelmoitavan ohjausyksikön suorittama ohjelma on sarja loogisia ykkösiä ja nollia, fysikaalisina suureina eri jännitetasoja, ohjausyksikön ohjelmamuistissa. Ohjausyksikkö tulkitsee ohjelmamuistin bitit konekäskyiksi. Konekäskyt ovat yksinkertaisia suorittimen operaatioita, kuten peruslaskutoimituksia, muistin luku- ja kirjoitusoperaatioita ja ohjelman suoritusjärjestystä ohjaavia hyppykäskyjä. Ohjausyksikön suoritin tuntee rajallisen määrän konekäskyjä, joista muodostuu sen konekieli. Ohjelmoija ei ohjausyksikköä ohjelmoidessaan asettele jännitetasoja yksittäisiin muistipaikkoihin, vaan hän kirjoittaa ohjelman valitsemallaan ohjelmointikielellä. Ohjelmointikieli on kompromissi suorittimen ymmärtämän konekielen ja ihmisten keskenään puhuman luonnollisen kielen välillä. Se on formaali kieli, jota ohjelmoijien on kohtuullisen helppo kirjoittaa ja lukea, ja joka voidaan yksiselitteisesti muuntaa suorittimen ymmärtämäksi konekieleksi ja kirjoittaa bitteinä ohjausyksikön ohjelmamuistiin. Tässä luvussa esitellään ohjelmoinnin historiaa sekä sulautettujen järjestelmien ohjelmoinnissa yleisesti käytettyjä ohjelmointikieliä ja -paradigmoja.

3.1 Ohjelmoinnin historia

Alkujaan tietokoneet ohjelmoitiin kirjoittamalla binäärikoodia, mikä vastasi täsmälleen suorittimen ymmärtämää konekieltä. Konekielisiä käskyjä yhdistelemällä saatiin aikaiseksi monimutkaisempia laskutoimituksia ja algoritmeja. Ohjelmoija kirjoitti binäärikoodin reikäkorteille tai reikänauhalle, jotka syötettiin tietokoneeseen, joka suoritti ohjelman välittömästi [4]. Tietokone ei tarkistanut binäärikoodia ennen sen suorittamista, joten yksikin väärään kohtaan lyöty reikä tai sen puuttuminen sai ohjelman toimimaan väärin, ja ohjelmoijan oli itse löydettävä ja korjattava virheensä, mikä pahimmillaan tarkoitti koko reikäkortin kirjoittamista uudelleen. Binäärikoodin kirjoittaminen oli siten äärimmäisen työlästä. Lisäksi binäärikoodia oli mahdotonta suorittaa uudelleen muun tyyppisellä suorittimella kuin mille se oli alun perin kirjoitettu, koska suorittimien ymmärtämä konekieli vaihtelee eri suorittimien välillä. Binäärikoodia kutsutaan myös ensimmäisen sukupolven ohjelmointikieleksi. [12]

Ohjelmointityötä helpottamaan kehitettiin symboliset konekielet, eli assembly-kielet, joita kutsutaan myös toisen sukupolven ohjelmointikieliksi. Assembly-kielissä kullakin konekielisellä käskyllä on selkokielinen symboli, joka tyypillisesti on lyhenne käskyä kuvaavasta englanninkielisestä sanasta. Ohjelmoija käyttää symbolia konekielisen käskyn binäärikoodin sijaan. Ennen kuin assembly-kielellä kirjoitettua ohjelmaa voidaan suorittaa, se täytyy syöttää ohjelmalle, jota kutsutaan kokoajaksi (assembler). Kokoaja muuntaa assembly-kielisen koodin vastaavaksi binäärikoodiksi, joka voidaan suorittaa. Kokoaja

tarkistaa myös, että sille syötetty koodi on syntaktisesti oikein, ja kaikki annetut symboliset konekäskyt ovat suorituskelpoisia sille suorittimelle, jolla ne on tarkoitus suorittaa. Tällä tavalla ohjelmakoodissa olevat kirjoitusvirheet huomataan ennen kuin ohjelmaa yritetään suorittaa. Kokoaja osaa usein myös ilmoittaa virheen täsmällisen paikan koodissa, mikä helpottaa virheen korjaamista. [12]

Aivan kuten binäärikoodi, assembly-koodi on tiukasti sidoksissa siihen suoritintyyppiin, jolle se on alun perin kirjoitettu. Assembly-kielisen ohjelman uudelleenkäyttö on mahdollista vain, jos uusi suoritintyyppi tukee vähintään samoja symbolisia konekäskyjä kuin alkuperäinen.

Vaikka assembly-kielet ovat saaneet väistyä korkean tason kielien tieltä pääasiallisina ohjelmointikielinä, niitä käytetään yhä nykyään tietyissä erikoistapauksissa. Assemblyn etuna korkean tason kieliin on se, että se mahdollistaa suoritinkohtaisten erikoiskäskyjen hyödyntämisen ja konekäskyjen suorituksen ajoittamisen kellojakson tarkkuudella. Näistä syistä assembly-kieliä käytetään nykyäänkin varusohjelmien ja käyttöjärjestelmien toteutuksessa. Näissäkin tapauksissa suurin osa koodista on yleensä kirjoitettu korkean tason kielellä, ja vain tarkkaa ajoitusta tai suorittimen erikoiskäskyjä vaativat osat on kirjoitettu suorittimen assembly-kielellä.

Assembly-kielen suurin vahvuus on myös sen suurin heikkous – assembly-koodia kirjoitetaan konekäsky kerrallaan, ja ohjelmistojen monimutkaistuessa ja koon kasvaessa ohjelmointiin vaadittavan työn määrä kasvoi kohtuuttoman suureksi. Ohjelmoijien tuottavuus ei pysynyt enää ohjelmistojen vaatimusten perässä. Tuottavuuden parantamiseksi kehitettiin kolmannen sukupolven ohjelmointikielet, eli korkean tason imperatiiviset kielet [12]. Korkean tason kielissä ohjelmoija ei enää kirjoita yksittäisiä konekäskyjä, vaan lauseita, jotka voivat kuvata useaa konekielistä käskyä. Ohjelmointikielen imperatiivisuus tarkoittaa, että ohjelmakoodia suoritetaan rivi kerrallaan siinä järjestyksessä kuin ohjelmoija on päättänyt. Korkean tason imperatiiviset kielet ovat rakenteisia, eli ne tarjoavat ohjelman suoritusjärjestyksen hallintaan auttavia rakenteita, kuten funktioita, ehtoja toistorakenteita. Assembly-kielissä ohjelmoijan piti ilmaista vastaavat rakenteet hyppykäskyillä, jotka tekivät ohjelman logiikan seuraamisesta ihmiselle vaikeaa.

Korkean tason kielissä ohjelmoija voi antaa muuttujille, funktioille ja luokille selkokieliset, niiden tarkoitusta vastaavat nimet. Korkean tason kielet ovat ihmiselle paljon luettavampia kuin aiempien sukupolvien ohjelmointikielet. Hyvin kirjoitettu korkean tason kieli muistuttaa jo läheisesti luonnollista kieltä, ja maallikkokin saattaa sitä ymmärtää. Taulukko 6 havainnollistaa korkean tason ohjelmointikieli C:n eroja ARMv7-suoritinarkkitehtuurille spesifiseen binäärikoodiin ja assemblyyn ymmärrettävyyden ja ilmaisuvoiman kannalta. Esimerkin binääri-, assembly- ja C-koodi toteuttavat saman logiikan. Binäärikoodi on tilan säästämiseksi esitetty heksadesimaalimuodossa.

Taulukko 6. *Binäärikoodi, assembly ja C.*

Ensimmäinen sukupolvi: Binäärikoodi (ARMv7)	**Toinen sukupolvi: Assembly (ARMv7)**	**Kolmas sukupolvi: C**

0x08301BE5
0x013083E2
0x08300BE5
	ldr r3,[sp]
	adds r3,r3,#1
	str r3,[sp]
		laskuri = laskuri + 1;

Korkean tason ohjelmointikielellä kirjoitettu ohjelma on käännettävä tai tulkattava konekielelle ennen kuin se voidaan suorittaa. Kääntäminen tarkoittaa prosessia, jossa korkean tason ohjelmakoodi muunnetaan kokonaisuudessaan konekielelle ennen ohjelman suorittamista. Kääntäminen tarvitsee tehdä vain kerran, ja ohjelmaa voi sen jälkeen suorittaa aina uudelleen ilman uudelleen kääntämistä. Kääntämisen suorittavaa ohjelmaa sanotaan kääntäjäksi. Kääntäjä tarkistaa myös, että ohjelmakoodi on syntaktisesti ja semanttisesti oikein, ja osaa varoittaa epäilyttävästä koodista, kuten alustamattomista muuttujista. Kääntäjä osaa myös optimoida tuotettavaa konekoodia, jolloin siitä tulee tehokkaampaa kuin mitä ihminen olisi osannut assemblyllä tai binäärikoodilla kirjoittaa. Esimerkkejä käännettävistä korkean tason kielistä ovat C ja C++.

Tulkkaamisessa korkean tason ohjelmakoodia muunnetaan konekielelle sitä mukaa kuin sitä suoritetaan. Tulkattavat ohjelmointikielet ovat yleensä skriptikieliä, eikä niitä käytetä sulautettujen järjestelmien toteuttamiseen suoritusaikaisesta tulkkaamisesta johtuvan huonon suorituskyvyn vuoksi.

Toisin kuin assembly, korkean tason kielet eivät ole sidoksissa suorittimen konekieleen, vaan korkean tason kielellä kirjoitutettu ohjelmakoodi voidaan sopivalla kääntäjällä tai tulkilla saada toimimaan millä tahansa suorittimella, mikä mahdollistaa ohjelmakoodin uudelleenkäytön sellaisenaan.

Kolmannen sukupolven ohjelmointikielten jälkeen on kehitetty yhä edistyneempiä ohjelmointikieliä. Neljännen sukupolven ohjelmointikieliksi kutsutaan deklaratiivisia ohjelmointikieliä, joissa ohjelmoijan ei tarvitse enää ottaa kantaa suoritusjärjestykseen, vaan ainoastaan kuvailla ohjelman tai algoritmin haluttu lopputulos [12]. Kääntäjän tai tulkin tehtäväksi jää ratkaista paras tapa halutun lopputuloksen saavuttamiseksi. Deklaratiiviset ohjelmointikielet helpottavat ohjelmointityötä, mutta ohjelmoijalla ei ole niissä enää valtaa vaikuttaa algoritmin toteutusyksityiskohtiin. Esimerkkejä neljännen sukupolven ohjelmointikielistä ovat funktionaalinen ohjelmointikieli Haskel ja tietokantakyselykieli SQL.

Viidennen sukupolven ohjelmointikielissä ohjelmoija määrittää rajoitteet, joiden perusteella tietokone keksii itsenäisesti algoritmin ongelman ratkaisemiseksi [12]. Viidennen sukupolven ohjelmointikielillä ei ole vielä nykyään käytännön sovelluksia, vaan ne ovat ainoastaan akateemisen tutkimuksen kohteita. Tietokoneet eivät vielä nykyään osaa omin

päin kehittää riittävän tehokkaita algoritmeja, jotta viidennen sukupolven ohjelmointikielten soveltaminen käytännössä olisi mielekästä.

3.2 Ohjelmointikielet koneenohjauksessa

Koneenohjauksen sovellusalueella on edistyneempien ohjelmointikielten kehityksestä huolimatta pitäydytty kolmannen sukupolven ohjelmointikielissä. Koneenohjauksessa on välttämätöntä hallita koneen toimintaa laitteistoläheisesti, ja ohjelman suoritusjärjestyksellä on yleensä suuri merkitys. Kolmannen sukupolven ohjelmointikielet tarjoat sopivan tasapainon ohjelmointityön tuottavuuden ja yksityiskohtien hallinnan välillä. Koneenohjausjärjestelmien ohjausyksiköiden laskentateho on rajallinen, ja järjestelmällä on kovat reaaliaikavaatimukset, minkä takia koneenohjauksessa suositaan käännettäviä ohjelmointikieliä niiden tehokkuuden vuoksi. Yleisiä koneenohjausjärjestelmien toteutuskieliä ovat C, C++ ja IEC 61131-3 -standardin määrittelemät ohjelmointikielet.

3.2.1 C ja C++

Dennis Ritchie kehitti C:n vuonna 1972, ja se on vielä nykyäänkin yksi maailman yleisimmin käytettyjä ohjelmointikieliä etenkin sulautettujen järjestelmien, käyttöjärjestelmien ja varusohjelmistojen toteutuksessa. C on korkean tason käännettävä, rakenteinen ja proseduraalinen ohjelmointikieli. C on suosittu laitteistoläheisessä ohjelmoinnissa, koska se tarjoaa ohjelmoijalle paljon valtaa matalan tason yksityiskohtiin olematta kuitenkaan sidoksissa suorittimen konekieleen. Samalla C tarjoaa riittävät työkalut korkeamman tason logiikan toteuttamiseen. C on UNIX-käyttöjärjestelmien toteutuskieli, mikä on mahdollistanut C:n suuren suosion myös PC-sovellusten toteutuksessa. [13]

C on standardoitu ohjelmointikieli, minkä ansiosta kielen ominaisuudet ja standardikirjastot ovat yksiselitteisesti määritellyt, eikä kielessä ole keskenään kilpailevia ja epäyhteensopivia murteita. C:n standardi on ISO/IEC 9899, joka on hyvin vakaa, ja muuttuu harvoin. Viimeisin standardi on vuodelta 2011, ja sen edeltäjä vuodelta 1999. [14]

C++ on C:n pohjalta kehitetty olio-ohjelmointikieli. Bjarne Stroustrup aloitti C++:n kehityksen vuonna 1979. Hän kutsui kieltä aluksi nimellä C with Classes (C Luokilla). Innoituksen uuden kielen kehittämiseen Stroustrup oli saanut työskenneltyään oliokeskeisen Simula 67 -kielen parissa. Hän halusi kehittää oliokeskeisen kielen, joka olisi yhtä helppokäyttöinen kuin Simula 67, mutta sisältäisi samat ominaisuudet ja olisi yhtä tehokas kuin C. Vuonna 1983 kielen nimeksi vaihdettiin C++. Nimi korostaa sitä, että C++ on laajennos C:stä. C++ on vielä nykyäänkin täysin yhteensopiva C:n kanssa, eli mikä tahansa C-ohjelma voidaan kääntää myös C++-kääntäjällä. [15]

C++ on C:n tavoin standardoitu kieli. C++:n standardi on ISO/IEC 1488, jota uudistetaan useammin kuin C:n standardia, jotta se pysyisi paremmin mukana uusien ohjelmointiparadigmojen ja ohjelmoijien tarpeiden perässä. C++:n viimeisin standardi on vuodelta

2017, ja sen edeltäjä vuodelta 2014. Seuraava standardin versio suunnitellaan julkaistavaksi vuonna 2020. [16]

C ja C++ ovat yleisiä ohjelmointikieliä muutenkin kuin sulautettujen järjestelmien toteutuksessa. Nämä kielet ovat ohjelmistoalalla hyvin yleisesti tunnettuja, joten kieliä ei esitellä tässä diplomityössä tarkemmin. Yksityiskohtaista tietoa kielistä on vapaasti saatavilla esimerkiksi cplusplus.com ja cppreference.com sivustoilta.

3.2.2 IEC 61131-3

IEC 61131-3 on kolmas osa IEC 61131 -standardia [17], joka käsittelee ohjelmoitavien ohjausyksiköiden ohjelmointia ja turvallisuutta. Standardin kolmas osa määrittelee viisi suositeltavaa ohjelmointikieltä ohjausyksiköiden ohjelmointiin. Nämä ohjelmointikielet ovat:

- Tekstimuotoinen, korkean tason imperatiivinen ST (Structured Text).
- Tekstimuotoinen, assemblyä muistuttava IL (Instruction List), joka on nykyään epäsuosiossa.
- Graafinen FBD (Function Block Diagram)
- Graafinen, vanhanaikaisia relekytkentäkaavioita muistuttava LD (Ladder Diagram)
- Ohjelman suoritusvuota määrittävä SFC (Sequential Function Chart).

Tässä diplomityössä käsitellään tarkemmin vain ST- ja FBD-kieliä, koska niitä on käytetty luvun 5 esimerkkijärjestelmässä.

IEC ei tarjoa valmista toteutusta standardin ohjelmointikielille, vaan niiden toteuttaminen on kolmansien osapuolten tarjoamien ohjelmointiympäristöjen vastuulla. Tunnetuin IEC 61131-3 -standardin toteuttava ohjelmointiympäristö on saksalaisen 3S Smart Software Solutions -yrityksen kehittämä CODESYS (COntroller DEvelopment SYStems). CODESYS tarjoaa IEC 61131-3 -ohjelmointikielten toteutuksen lisäksi kehitystyötä helpottavia valmiita kirjastoja, integroidun kehitysympäristön (IDE, Integrated Development Environment) ja ajonaikaisen ympäristön CODESYS-sovellusten laitteelle lataukseen, suorittamiseen ja debuggaukseen. CODESYS-sovellukset ovat alustariippumattomia, ja CODESYS:n ajonaikainen ympäristö on saatavilla suurelle joukolle ohjausyksiköitä, mikä helpottaa koodin uudelleenkäyttöä ja pienentää riippuvuutta PLC-valmistajasta. [18]

CODESYS-ympäristöstä on olemassa useita eri versioita, joiden ominaisuudet vaihtelevat merkittävästi keskenään. Sandvikin vanhoissa projekteissa on käytetty CODESYS 2.3 tai vanhempaa versiota, ja uudemmissa 3.5-versiota. Merkittävin ero versioiden välillä on 3.5-version mahdollistama olio-ohjelmointi IEC 61131-3 -ohjelmointikielillä. [18].

IEC 61131-3 -standardi määrittelee kaikille ohjelmointikielille yhteiset perustietotyypit, jotka on listattu taulukossa 7. Numeeriset perustietotyypit on listattu standardin määrittelemän tietotyypin koon mukaan pienimmästä suurimpaan. Ajan ja merkkijonojen toteutusten koot riippuvat standardin toteutuksesta. Ohjelmoija voi perustietotyyppien lisäksi määritellä omia tietotyyppejä, kuten luetteloja, tietueita, taulukoita ja arvoalueita. [19, s. 67-98] Olio-ohjelmoinnin mahdollistava CODESYS 3.5 mahdollistaa myös abstraktien tietotyyppien (rajapintojen) määrittelyn.

Taulukko 7. *IEC 61131-3 perustietotyypit [19, s. 67-98].*

Bittijonot	**Etumerkilliset kokonaisluvut**	**Etumerkittömät kokonaisluvut**	**Liukuluvut**	**Aika, päiväys, merkkijonot**
BOOL BYTE WORD DWORD LWORD	SINT INT DINT LINT	USINT UINT UDINT ULINT	REAL LREAL	TIME DATE TIME_OF_DAY DATE_AND_TIME STRING

IEC 61131-3 -standardi määrittelee sovellusten rakenteen. Standardin mukainen sovellus koostuu yksiköistä, joista käytetään nimeä POU (Program Organization Unit). POU-tyyppejä ovat funktiot, funktiolohkot ja ohjelmat, joita yhdistelemällä muodostuu lopullinen suoritettava sovellus [19, s. 21-65]. Kukin POU voidaan toteuttaa millä tahansa viidestä ohjelmointikielestä, ja eri kielillä kirjoitettuja POU:ita on mahdollista yhdistellä keskenään.

Funktiot ovat yksinkertaisin POU-tyyppi. Funktio on sarja komentoja, ja sillä ei ole säilyvää sisäistä tilaa. Funktiolle voidaan määrittää sisäänmenoparametreja, joita se käyttää laskentansa lähtötietoina. Funktiolla on yksi paluuarvo ja vapaavalintainen määrä ulostuloparametreja, joihin funktio sijoittaa laskentansa tulokset. Funktiolla voi olla lisäksi sisäisiä muuttujia, joita se voi käyttää välitulosten tallentamiseen. Sisäiset muuttujat kuitenkin alustetaan uudelleen jokaisella funktion kutsukerralla, ja niille varattu muisti vapautetaan funktion suorituksen päätyttyä. Edellisillä funktion suorituskerroilla tallennetut välitulokset eivät siten säily. Funktion parametrit, paluuarvo ja sisäiset muuttujat voivat olla mitä tahansa standardin määrittelemiä perustietotyyppejä tai ohjelmoijan itse määrittelemiä tietotyyppejä. Funktio voi sisäisessä toteutuksessaan kutsua muita funktioita, mutta ei muita POU-tyyppejä tai rekursiivisesti itseään. [19, s. 21-65]

Funktiolohkot muistuttavat funktioita, ja niillä on samaan tapaan sisäänmeno- ja ulostuloparametreja ja sisäisiä muuttujia, mutta ei paluuarvoa. Toisin kuin funktioiden, funktiolohkojen sisäiset muuttujat säilyvät funktiolohkon suorituskertojen välillä. Funktiolohkosta on siksi luotava instanssi ohjausyksikön muistiin ennen kuin funktiolohkoa voi käyttää. Olio-ohjelmointia tukevassa CODESYS 3.5 -ohjelmointiympäristössä funktiolohkot vastaavat luokkia, ja funktiolohkojen instanssit olioita. Funktiolohkoille voi

CODESYS 3.5 -ympäristössä määritellä metodeja, ja ne voivat toteuttaa abstrakteja rajapintoja ja periytyä toisista funktiolohkoista. Funktiolohkot voivat koostua muista funktiolohkoista, ja ne voivat käyttää mitä tahansa funktioita sisäisessä toteutuksessaan. [19, s. 21-65]

Ohjelmat ovat kaikkein korkeimman tason POU-tyyppi. IEC 61131-3 -sovellus koostuu vähintään yhdestä ohjelmasta (moniajoon kykenevillä ohjausyksiköillä voi olla useita rinnakkain suoritettavia ohjelmia), joka toimii sovelluksen aloituspisteenä. Ajonaikainen ympäristö suorittaa ohjelman syklisesti. Muut POU:t eivät voi suorittaa ohjelmaa. Sen sijaan ohjelma voi käyttää toteutuksessaan mitä tahansa funktiolohkoja tai funktioita. Ohjelma muistuttaa siten rakenteeltaan ja toiminnaltaan funktiolohkoa. Ohjelma voi funktiolohkosta poiketen määritellä globaaleja muuttujia, ja se voi lukea ja ohjata ohjausyksikön IO-rajapintaa. [19, s. 21-65]

Ohjelmissa Ohjelma 1 ja Ohjelma 2 on esimerkki IEC-61131-3 -standardin ohjelmointikielillä toteutetuista funktiolohkoista. Ohjelma 1 on kirjoitettu ST-kielellä, ja toteuttaa kuvitteellisen puominohjauslogiikan. Ohjelma 2 on kirjoitettu FBD-kielellä, ja näyttää, miten ohjelman 1 funktiolohkoa voidaan käyttää FBD-kielisessä POU:ssa huolimatta eri toteutuskielestä.

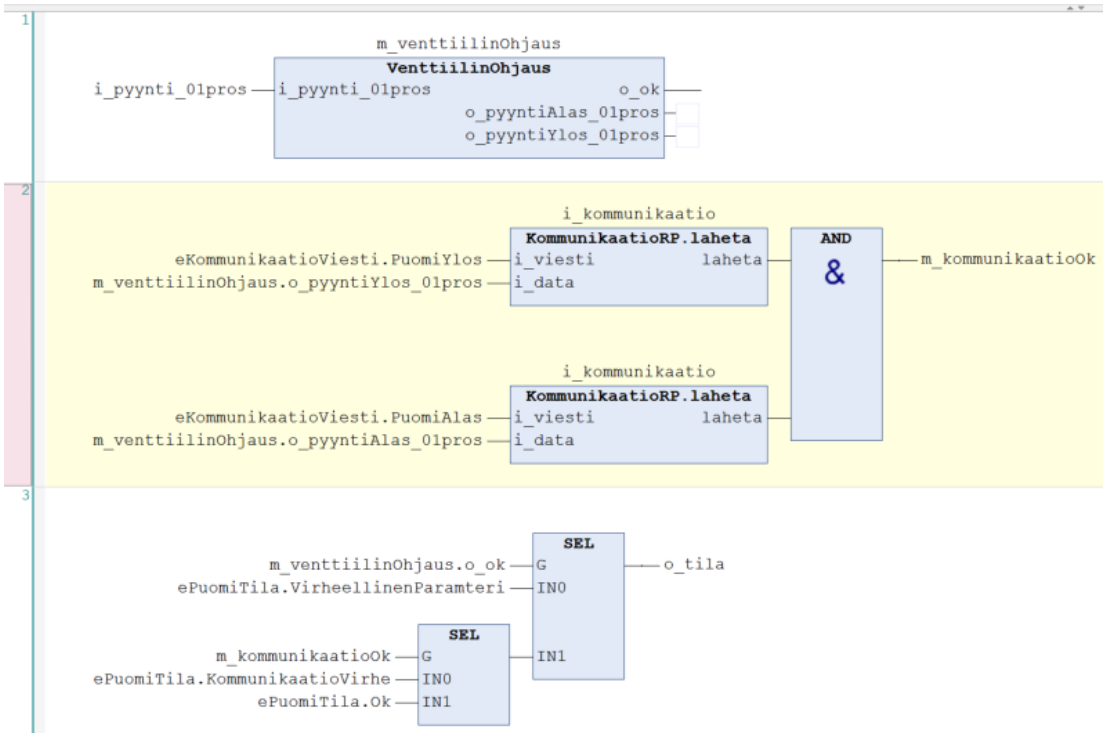
Esimerkit näyttävät myös, miten POU koostuu esittelyosasta ja toteutusosasta. Esittelyosa on aina tekstimuotoinen, ja se määrittelee POU:n sisäänmeno- ja ulostuloparametrit, sisäiset muuttujat, ja CODESYS 3.5:n tapauksessa myös toteutetut rajapinnat ja perityt funktiolohkot. Esittelyosa on samanlainen riippumatta POU:n toteutuskielestä. Toteutusosa määrittelee POU:n logiikan. Toteutusosa voi olla tekstimuotoinen tai graafinen riippuen POU:n toteutuskielestä. Molemmat esimerkit kuvaavat funktiolohkoa, mutta funktioilla ja ohjelmilla on täsmälleen sama rakenne [19, s. 21-65].

```
FUNCTION_BLOCK VenttiilinOhjaus
(* Muuntaa kaksisuuntaisen nopeuspyynnin venttiilinohjauspyynneiksi *)
VAR_INPUT
 i_pyynti_01pros : INT; (* Nopeuden pyynti ja suunta 0,1 prosentteina *)
END_VAR
VAR_OUTPUT
 o_pyyntiYlos_01pros : UINT := 0; (* Ylös-venttiilin ohjauspyynti *)
 o_pyyntiAlas_01pros : UINT := 0; (* Alas-venttiilin ohjauspyynti *)
 o_ok : BOOL := FALSE; (* Tosi, jos parametrit ovat valideja. *)
END_VAR
VAR CONSTANT
 MIN_PYYNTI : INT := -1000;
 MAX_PYYNTI : INT := 1000;
END_VAR

IF i_pyynti_01pros < MIN_PYYNTI OR i_pyynti_01pros > MAX_PYYNTI THEN
 o_ok := FALSE;
 o_pyyntiYlos_01pros := 0;
 o_pyyntiAlas_01pros := 0;
ELSE
 o_ok := TRUE;
```

```
 o_pyyntiYlos_01pros:=INT_TO_UINT(SEL(i_pyynti_01pros>0, i_pyynti_01pros,0));
 o_pyyntiAlas_01pros:=INT_TO_UINT(SEL(i_pyynti_01pros<0,-i_pyynti_01pros,0));
END_IF

END_FUNCTION_BLOCK
```

Ohjelma 1. *Esimerkki IEC-61131-3 ST-kielellä kirjoitetusta funktiolohkosta.*

```
FUNCTION_BLOCK Puomi
(* Ohjaa puomia operaattorin pyynnön mukaan *)
VAR_INPUT
 i_pyynti_01pros : INT; (* Nopeuden pyynti ja suunta 0,1 prosentteina *)
 i_kommunikaatio : KommunikaatioRP; (* Kommunikaatiorajapinta *)
END_VAR
VAR_OUTPUT
 o_tila : ePuomiTila; (* Puomin nykyinen tila *)
END_VAR
VAR
  m_venttiilinOhjaus : VenttiilinOhjaus; (* Ohjauslogiikan toteuttava FB *)
  m_kommunikaatioOk : BOOL; (* Välitulos kommunikaation onnistumisesta *)
END_VAR
```




```
END_FUNCTION_BLOCK
```

Ohjelma 2. *Esimerkki IEC-61131-3 FBD-kielellä kirjoitetusta funktiolohkosta piirrettynä CODESYS 3.5 -ohjelmointiympäristössä.*

3.3 Ohjelmointiparadigmat

Tässä aliluvussa käsitellään keskeisimpiä ohjelmointiparadigmoja, ja miten niitä voidaan soveltaa koneenohjausjärjestelmien toteuttamiseen. Ohjelmointiparadigmat ovat käytettävästä ohjelmointikielestä riippumattomia periaatteita, joiden mukaisesti ohjelmistoja

voidaan suunnitella ja toteuttaa. Tässä diplomityössä keskitytään kolmeen keskeisimpään paradigmaan: proseduraaliseen ohjelmointiin, olio-ohjelmointiin ja funktionaaliseen ohjelmointiin.

Vaikka ohjelmointiparadigmat eivät riipu ohjelmointikielestä, eri kielet tarjoavat vaihtelevasti työkaluja eri paradigmojen käyttämiseen. Taulukossa 8 on esitetty, miten hyvin C, C++ ja IEC 61131-3 CODESYS-toteutukset tukevat kutakin ohjelmointiparadigmaa. Taulukossa arvosana ’hyvin’ tarkoittaa, että ohjelmointikieli tarjoaa kaikki tarvittavat työkalut ohjelmointiparadigman toteuttamiseksi. Arvosana ’soveltaen’ tarkoittaa, että ohjelmointikieli mahdollistaa paradigman toteutuksen, mutta se vaatii erityistä vaivannäköä ja kuria ohjelmoijalta. Arvosana ’heikosti’ tarkoittaa, että ohjelmointikieli ei tarjoa tukea paradigman toteutukselle, tai se vaatisi kohtuutonta vaivannäköä ohjelmoijalta. Ohjelmointikielille annettuja arvosanoja on perusteltu tarkemmin seuraavissa aliluvuissa.

Taulukko 8. *Ohjelmointikielten tarjoama tuki ohjelmointiparadigmoille.*

Ohjelmointikieli	Proseduraalinen ohjelmointi	Olio-ohjelmointi	Funktionaalinen ohjelmointi
C	Hyvin	Soveltaen	Heikosti
C++	Hyvin	Hyvin	Soveltaen
IEC 61131-3	Hyvin	CODESYS 2.3: Soveltaen CODESYS 3.5: Hyvin	Heikosti

3.3.1 Proseduraalinen ohjelmointi

Proseduraalinen ohjelmointi on käsiteltävistä paradigmoista vanhin, ja juontaa juurensa rakenteisten ohjelmointikielten kehityksestä. Proseduraalisen paradigman mukaisesti toteutettu ohjelma koostuu proseduureista, jotka ohjelmointikielestä riippuen voivat tarkoittaa aliohjelmia, funktioita tai muita ohjelmointikielen tarjoamia rakenteita. Proseduuri on joukko loogisesti yhteenkuuluvia komentoja, jotka suoritetaan aina kutsuttaessa kyseistä proseduuria. Proseduurilla on yksiköllinen nimi, jonka avulla kutsuminen tapahtuu. Proseduurikutsun yhteydessä proseduurille voi yleensä välittää sen tarvitsemat lähtötiedot parametreina. Proseduraalisessa ohjelmoinnissa tiettyä proseduuria voidaan kutsua mistä tahansa osasta ohjelmaa, joskin ohjelmointikieli voi asettaa tälle rajoitteita tai lisävaatimuksia. Proseduuri on täten pala uudelleenkäytettävää koodia, ja ohjelman suorituksen aikana usein toistettavista komentosarjoista kannattaa tehdä proseduuri. Loogisesti yhteenkuuluvat proseduurit kannattaa koota samaan moduuliin, jos ohjelmointikieli sitä tukee.

Proseduraalisen ohjelmoinnin etuja ovat proseduurien uudelleenkäytön lisäksi laitteistoläheisyys. Tietokoneiden suorittimet tukevat luonnostaan proseduraalista paradigmaa. Proseduurin kutsu kääntyy prosessorin konekielessä yhdeksi hyppykäskyksi, ja proseduu-

rin sisäisesti käyttämä muisti varataan proseduurin suoritushetkellä pinosta, mikä on laitteiston kannalta yksinkertainen ja turvallinen tapa varata mustia. Proseduraalinen paradigma on näistä syistä yhä suosittu sulautettujen järjestelmien toteutuksessa.

Proseduurisen paradigman heikkous on proseduurien rajoittamattomat näkyvyysalueet ja ohjelman tilan käsittely. Rajoittamattomat näkyvyysalueet sallivat proseduurin kutsumisen mistä tahansa osaa ohjelmaa, milloin tahansa ohjelman suorituksen aikana. Proseduraalista paradigmaa laiskasti tai huolimattomasti käyttämällä syntyy niin kutsuttua spagettikoodia, jossa ohjelman suoritus hyppelee ristiin rastiin moduulista toiseen, eikä ohjelmalla ole selkeää rakennetta. Spagettikoodissa ohjelman logiikan seuraaminen ja suuren kokonaisuuden hahmottaminen on ihmiselle vaikeaa, ja koodin ylläpito ja jatkokehittämien on siten työlästä ja kallista. Spagettikoodin välttämiseksi ohjelmoijan on suunniteltava ohjelmalle selkeä rakenne, ja pitäydyttävä siinä.

Proseduraalinen paradigma ei tarjoa yksinkertaista ja ongelmatonta tapaa hallita ohjelman tilaa. Ohjelman tilalla tarkoitetaan tietoa, joka muuttuu ohjelman suorituksen kuluessa ja vaikuttaa ohjelman logiikkaan. Kaivoslastauskoneen koneenohjausjärjestelmässä tila voi olla esimerkiksi operaattorin valitsema vaihde, joka muuttuu operaattorin painaessa ohjaussauvan painikkeita, ja vaikuttaa muun muassa vaihdelaatikon ohjaukseen. Tieto valitusta vaihteesta on säilöttävä siten, että se on saatavilla koko ohjelman suoritusajan. Proseduraalisissa ohjelmointikielissä, kuten C:ssä, ainoa käytännöllinen ratkaisu on tallettaa tieto globaaliin muuttujaan, mitä yleensä pidetään hyvän ohjelmointitavan vastaisena. Globaalit muuttujat huolimattomasti käytettyinä vaikuttavat ohjelman logiikkaan ”salaa” koodin satunnaiselta lukijalta, mikä vaikeuttaa koodin ymmärrettävyyttä, ja siten ylläpitoa ja jatkokehitystä. Globaalien muuttujien aiheuttamat haitat voidaan pitää kohtuullisina huolellisella moduulisuunnittelulla.

C, C++ ja IEC-61131-3 tukevat hyvin proseduraalista ohjelmointiparadigmaa. Kukin ohjelmointikieli tarjoaa proseduraalisen paradigman noudattamiseen tarvittavat työkalut, eli rakenteisen suoritusvuon ja näkyvyysalueet, funktiot ja globaalit muuttujat. Näistä ohjelmointikielistä etenkin C:tä käytetään yleensä proseduraalisen paradigman mukaisesti.

3.3.2 Olio-ohjelmointi

Olio-ohjelmointi kehitettiin ratkaisuksi proseduraalisessa ohjelmointiparadigmassa hankalaan tiedon säilömiseen ja kapselointiin. Samoin kuin proseduraalisessa ohjelmoinnissa, ohjelma on yleensä jaettu moduuleihin, jotka sisältävät toteutuksen jollekin ohjelman loogiselle osakokonaisuudelle. Olio-ohjelmoinnissa kukin moduuli määrittelee tyypillisesti yhden luokan, joka vastaa moduulin logiikan toteutuksesta. Luokka on yhdistelmä palveluita, sekä tietoa, jota nämä palvelut toteutuksessaan käsittelevät. Luokan palveluita kutsutaan sen rajapinnaksi, ja luokan sisältämää tietoa sen tilaksi. [20]

Luokasta voidaan luoda ohjelman suoritusaikana useita instansseja, joita kutsutaan olioiksi. Kukin luokasta luotu olio sisältää saman rajapinnan ja sen toteutuksen, kuin kaikki muut saman luokan oliot, mutta joka oliolla on yksilöllinen, toisista olioista riippumaton tila. Luokan rajapinta on julkinen, ja ohjelman muut osat käyttävät luokan palveluita sen kautta. Hyvän ohjelmointitavan mukaisesti luokan tila sen sijaan on yksityinen, eli vain olio itse voi muokata omaa tilaansa. Muut oliot voivat vaikuttaa olion tilaan vain epäsuorasti käyttämällä sen rajapinnan palveluita. Tätä kutsutaan tiedon kapseloinniksi. Kapselointi estää muita olioita muokkaamasta kapseloitua tietoa asiattomalla tavalla, ja mahdollistaa tilan toteutuksen muuttamisen ilman, että se vaatisi muutoksia ohjelman muihin osiin. Kapselointi on merkittävä parannus proseduraaliseen paradigmaan nähden, jossa tiedon kurinalainen käsittely on yksin ohjelmoijan vastuulla. [20]

Olio-ohjelmoinnin ohjelmiston uudelleenkäytettävyyttä parantava erityispiirre on luokkien periytyminen. Olio-ohjelmointia tukevissa kielissä luokka voi periä toisen luokan, jolloin sillä on sama rajapinta ja sama rajapinnan toteutus, kuin perityllä luokalla. Perittyä luokkaa kutsutaan kantaluokaksi ja perivää luokkaa aliluokaksi. Aliluokka voi laajentaa omaa rajapintaansa uusilla palveluilla, tai se voi muokata perittyjen palvelujen toteutusta. Periytyminen mahdollistaa siten kantaluokan ohjelmakoodin uudelleenkäytön perityissä luokissa.

Luokat voivat toteuttaa myös abstrakteja rajapintoja. Luokan toteuttaessa abstraktin rajapinnan, se perii vain rajapinnassa määritellyt palvelut, mutta ei lainkaan toteutusta. Luokan on itse tarjottava toteutus kaikille toteutetun rajapinnan palveluille. Abstraktit rajapinnat mahdollistavat rajapinnan toteutuksen korvaamisen täysin eri toteutuksella, ilman että rajapinnan palveluita käyttäviä ohjelman osia tarvitsee muuttaa.

Luokkien periytyminen ja abstraktien rajapintojen toteuttaminen mahdollistavat polymorfismin eli monimuotoisuuden. Polymorfismilla tarkoitetaan olio-ohjelmoinnin kontekstissa sitä, että funktio tai muu vastaava kielen rakenne saadaan toimimaan eri tavalla riippuen sen parametrien todellisista tyypeistä. Staattisesti tyypitetyissä ohjelmointikielissä rajoituksena on, että vaihtoehtoisilla tyypeillä on yhteinen kantaluokka tai abstrakti rajapinta. Ohjelman 3 esimerkki selventää polymorfismia. Esimerkissä Jarru-funktiolohko käyttää RamppiRp-rajapintaa. RamppiRp-rajapinnasta on kaksi erilaista toteutusta, ja samaa Jarru-oliota voidaan käyttää tilanteen mukaan eri rampilla. Erilaisia RamppiRp-rajapinnan toteutuksia voidaan tehdä rajoittamaton määrä, eivätkä ne vaadi muutoksia Jarru-luokan toteutusukseen. Polymorfismi mahdollistaa täten abstraktia rajapintaa käyttävän Jarru-luokan uudelleenkäytön eri tilanteissa.

```
// Abstraktin RamppiRP-rajapinnan määrittely
INTERFACE RamppiRP
    METHOD paivita : INT
    VAR_INPUT
        i_arvo : INT;
    END_VAR
    END_METHOD
```

```
END_INTERFACE

// Funktiolohko, joka käyttää RamppiRp-rajapintaa:
FUNCTION_BLOCK Jarru
VAR_INPUT
    i_ramppi : RamppiRP;
END_VAR
VAR_OUTPUT
    o_pyynti_pros : INT;
END_VAR
...
o_pyynti_pros := LIMIT(0, i_ramppi.paivita(o_pyynti_pros), 100);
...
END_FUNCTION_BLOCK

// RamppiRP-rajapinnan kaksi vaihtoehtoista toteutusta
FUNCTION_BLOCK TasainenRamppi IMPLEMENTS RamppiRP
VAR_INPUT
    i_kulmakerroin : INT;
END_VAR
...
END_FUNCTION_BLOCK

FUNCTION_BLOCK KiihtyvaRamppi IMPLEMENTS RamppiRP
VAR_INPUT
    i_kiihtyvyys : INT;
END_VAR
...
END_FUNCTION_BLOCK

// Koodi, joka käyttää Jarrua kahdella eri rampilla
PROG Paaohjelma
VAR
  jarru : Jarru;
  ramppi1 : TasainenRamppi;
  ramppi2 : KiihtyvaRamppi;
END_VAR
...
ramppi1.i_kulmakerroin := 42;
jarru(i_ramppi := ramppi1);
...
ramppi2.i_kiihtyvyys := 19;
jarru(i_ramppi := rammpi2);
...
END_PROG
```

Ohjelma 3. *CODESYS 3.5 -esimerkki polymorfismista.*

Olio-ohjelmoinnin on mainostettu ratkaisevan ohjelmiston uudelleenkäyttöongelman erityisesti periytymisen ansiosta [21], mutta käytännön tulokset eivät ole vastanneet odotuksia [22]. Biddle ja Tempero esittävät artikkelissaan Inheritance and Reusability [22], että periyttäminen parantaa ohjelmiston uudelleenkäytettävyyttä, mutta periyttämistä käytetään usein väärin tässä tarkoituksessa. Biddlen ja Temperon mukaan periyttämisen käyttäminen on järkevää vain polymorfismin toteuttamiseksi. Jos kaksi eri luokkaa muuten tarvitsevat samaa koodia toteutuksessaan, periyttämisen sijaan on järkevämpää koostaa nämä kaksi luokkaa kolmannesta luokasta, joka toteuttaa yhteisen toiminnallisuuden.

Olioparadigma on verrattain uusi sulautettujen järjestelmien toteutuksessa. Olio-ohjelmoinnissa käytetyt abstraktiokerrokset kasvattavat suoritettavan ohjelman kokoa ja vaativat ylimääräistä suoritusaikaa. Sulautetuissa järjestelmissä suoritustehoa ja muistia on rajoitetusti, minkä vuoksi myös dynaaminen muistinvaraus on tabu, mikä vaikeuttaa olioparadigman soveltamista. Ohjausyksiköiden suorituskyvyn ja mustin määrän kasvaessa olio-ohjelmoinnista on tullut mahdollinen paradigma myös sulautetuissa järjestelmissä. Olioparadigma on ihmiselle helpommin ymmärrettävä kuin proseduraalinen paradigma, ja se tarjoaa enemmän mahdollisuuksia ohjelmiston abstrahointiin ja uudelleenkäyttöön. Olioparadigma kasvattaa täten työn tuottavuutta, minkä vuoksi sen suosio kasvaa myös sulautetuissa ohjelmistoissa. Ymmärrettävyytensä vuoksi olio-ohjelmointia suositellaan käytettäväksi etenkin koneiden turvallisuuskriittisessä ohjelmistossa [24].

Tässä työssä erityiseen tarkasteluun valituista ohjelmointikielistä C++ ja IEC 61131-3 CODESYS 3.5 -ohjelmointiympäristössä tukevat olio-ohjelmointia hyvin. Ne tarjoavat kaikki tarvittavat työkalut paradigman soveltamiseen, kuten kapseloinnin, periyttämisen, abstraktit rajapinnat ja polymorfismin. Molempia kieliä on siitä huolimatta mahdollista käyttää puhtaasti proseduraalista paradigmaa noudattaen.

C ja IEC 61131-3 CODESYS 2.3 -ohjelmointiympäristössä tukevat olio-ohjelmointia soveltaen. Ne eivät tarjoa periyttämistä tai abstrakteja rajapintoja. Paradigmaa on silti mahdollista soveltaa rajoitetussa määrin. C ei tarjoa apua tiedon kapselointiin, mutta ohjelmoija voi itse pidättäytyä muokkaamasta kapseloitavaa tietoa muuten kuin tiettyjen proseduurien kautta. C:ssä polymorfismia voi soveltaa funktio-osoittimien avulla, mutta se on paljon työläämpää kuin olio-ohjelmointia täysin tukevissa kielissä. IEC 61131-3 CODESYS 2.3 -toteutus tukee tiedon kapselointia funktiolohkoissa, ja mahdollistaa funktiolohkojen koostamisen toisista funktiolohkoista, mutta funktio-osoittimia vastaavan mekanismin puuttuessa polymorfismin soveltaminen ei onnistu.

3.3.3 Funktionaalinen ohjelmointi

Funktionaalinen ohjelmointiparadigma perustuu lambdakalkyyliin, joka on matematiikassa formaali tapa esittää laskentaa funktioiden avulla. Funktionaalisen ohjelmointiparadigman keskeisiä ominaisuuksia ovat tiedon muuttumattomuus, puhtaat funktiot, funktioiden koostaminen ja korkeamman tason funktiot. [23, s. 167-190]

Tiedon muuttumattomuus tarkoittaa, että ohjelma ei voi muokata muuttujia tai tietorakenteita niiden luomisen jälkeen. Olemassa olevan tiedon muokkaamisen sijaan alkuperäisestä muuttujasta tai tietorakenteesta tehdään kopio, johon halutut muutokset tehdään. Tiedon muuttumattomuus tekee ohjelman rinnakkaisuuden toteuttamisen helpoksi, koska muuttuvan datan suojaamiseksi ei tarvita lukituksia. Tiedon muuttumattomuus huonontaa kuitenkin ohjelman suorituskykyä ja vaatii enemmän muistia. Pienikin muutos suuren tietorakenteen yhteen alkioon pakottaa kopioimaan koko alkuperäisen tietorakenteen.

Puhtaan funktionaaliset ohjelmointikielet soveltuvat siksi huonosti laitteistoresursseiltaan rajallisten sulautettujen järjestelmien toteutukseen.

Puhtailla funktioilla ei ole sivuvaikutuksia, ja niiden tulos riippuu vain ja ainoastaan funktiokutsun parametreista. Puhtaat funktiot palauttavat samoilla parametreilla aina saman arvon riippumatta kutsuhistoriasta tai muista suoritetuista funktioista. Puhtaat funktiot tekevät ohjelmasta ymmärrettävämmän, koska puhtaat funktiot eivät voi salaa sivuvaikutuksenaan muokata ohjelman globaalia tilaa, eikä globaali tila voi vaikuttaa puhtaan funktion toimintaan. Puhtaat funktiot tekevät samasta syystä ohjelmasta myös helposti testattavan. Helpon testattavuuden ja ymmärrettävyyden vuoksi funktioiden puhtauteen ja sivuvaikutusten välttämiseen kannattaa pyrkiä, vaikka ohjelma ei yleisesti noudattaisikaan funktionaalista paradigmaa. Näin on etenkin turvallisuuskriittisen ohjelmiston kehityksessä, jossa ohjelmakoodin ymmärrettävyys ja kattava testaaminen ovat pakollisia [24].

Funktionaalisessa paradigmassa funktiot ovat ensimmäisen luokan kansalaisia, eli niitä käsitellään ikään kuin dataa: funktion voi sijoittaa muuttujaan ja sen voi antaa parametrina toiselle funktiolle [23, s. 167-190]. Funktiota, joka ottavat parametrinaan toisen funktion, kutsutaan korkeamman tason funktioksi. Funktionaalisissa ohjelmointikielissä funktioita voi usein myös koostaa toisistaan, eli luoda uusi funktio, joka käyttää ensimmäisen funktion paluuarvoa toisen funktion parametrina ja suorittaa molemmat. Korkeamman tason funktiot tukevat ohjelmiston uudelleenkäyttöä samaan tapaan kuin polymorfismi olio-ohjelmoinnissa. Funktioiden koostaminen mahdollistaa funktioiden uudelleenkäytön uusien funktioiden raaka-aineina.

Moderni C++ (C++11 tai uudempi standardi) mahdollistaa melko täsmällisen funktionaalisen paradigman noudattamisen. Uudemmat C++-standardit tukevat anonyymejä lambdafunktioita ja funktio-olioita eli funktoreita, jotka muistuttavat läheisesti funktionaalisen ohjelmoinnin ensimmäisen luokan kansalaisina pidettäviä funktioita. Funktoreita ja lambdafunktioita voi välittää parametreina muille funktioille, ja niistä voi luoda uuden tyyppisiä funktioita parametrien sidonnalla. C++-kääntäjä valvoo funktioiden puhtautta ja tiedon muuttumattomuutta, jos niiden määrittelyssä on käytetty const-avainsanaa. Vanhemmissakin C++-standardeissa geneerisellä metaohjelmoinnilla on mahdollista jäljitellä funktionaalista ohjelmointiparadigmaa, mutta metaohjelmoinnin vaikealukuisen syntaksin vuoksi se ei ole käytännössä järkevää. [23, s. 167-190]

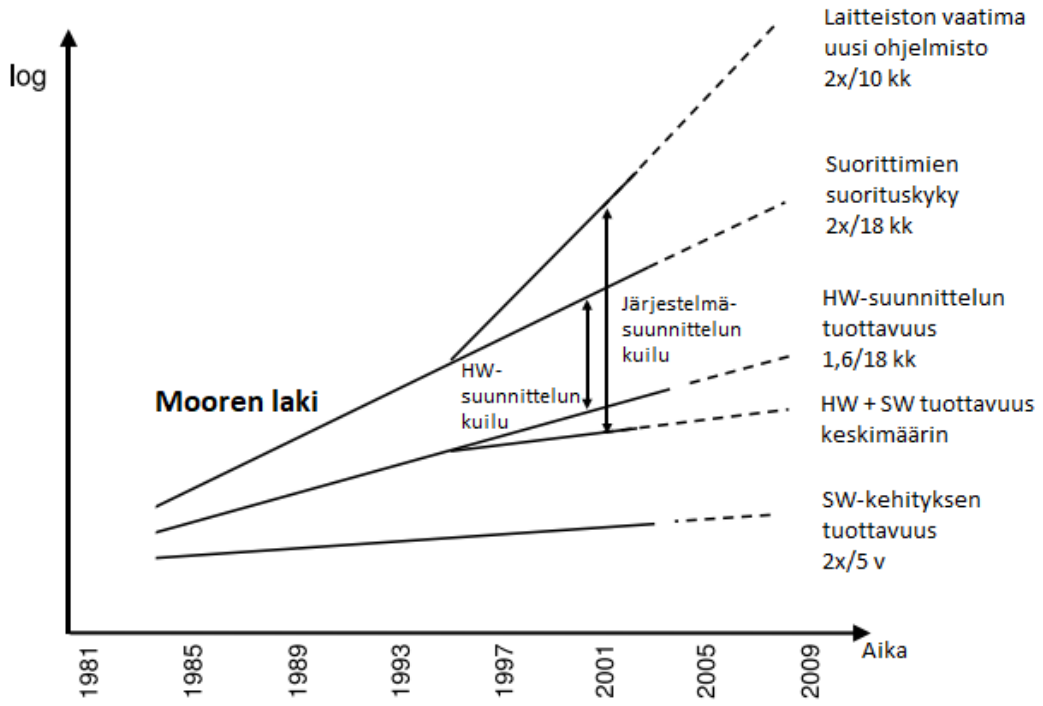
C:ssä ja IEC-61131-3 -ohjelmointikielissä funktionaalista paradigmaa voi soveltaa hyvin rajoitetusti. Molemmissa kielissä on mahdollista tehdä puhtaita funktioita, joilla ei ole sivuvaikutuksia, mutta paradigmassa pitäytyminen on ohjelmoijan omalla vastuulla. C:ssä funktio-osoittimia voi välittää parametreina muille funktioille, mutta funktioiden koostaminen tai parametrien sitominen eivät ole mahdollisia. IEC-61131-3 -ohjelmointikielissä ei ole funktio-osoittimia vastaavaa mekanismia, joten korkeamman tason funktiot tai funktioiden koostaminen eivät ole mahdollisia.

4. OHJELMISTON UUDELLEENKÄYTTÖ

Tässä luvussa perustellaan ohjelmiston uudelleenkäytön tarve ja esitellään keskeiset menetelmät uudelleenkäytettävän ohjelmiston suunnitteluun ja systemaattiseen uudelleenkäyttöön.

4.1 Miksi ohjelmistoa on käytettävä uudelleen?

Mooren lain mukaan yhdelle puolijohdesirulle integroitavien transistorien lukumäärä kaksinkertaistuu 18 kuukaudessa [25, s. 1-13]. Käytännössä tämä tarkoittaa suorittimien suorituskyvyn eksponentiaalista kasvua. Suorituskyvyn kasvu on mahdollistanut yhä monimutkaisempien ohjelmistojen toteuttamisen. Valitettavasti ohjelmoijien tuottavuus ei ole pysynyt ohjelmistojen kasvavien vaatimusten perässä. Ohjelmistokehityksen tuottavuuden arvioidaan kaksinkertaistuvan viidessä vuodessa [25, s. 1-13], eli huomattavasti hitaammin kuin suorittimien suorituskyky. Laitteistosuunnittelu ei sekään ole pysynyt teknologian kehityksen perässä. Kuvassa 10 on esitetty suorittimien suorituskyvyn ja niitä hyödyntävien järjestelmien kehityksen välinen kuilu.



Kuva 10. *Järjestelmäsuunnittelun tuottavuuskuilu [25, s. 1-13].*

Ratkaisu tuottavuuskuilun pienentämiseen on ohjelmiston uudelleenkäyttö. Uuden järjestelmän ohjelmiston kehittäminen tyhjästä on kohtuuttoman työlästä, ja alun perin käytettäväksi suunniteltu laitteisto on auttamatta vanhentunut ohjelmistokehityksen valmistuttua. Olemassa olevan ohjelmiston uudelleenkäyttö pienentää huomattavasti uuden järjestelmän kehitykseen vaadittavaa työtä, ja järjestelmä voidaan saattaa markkinoille kohtuullisessa ajassa.

Yrityksen kyky ohjelmiston uudelleenkäyttöön vaikuttaa suoraan sen markkina-asemaan. Mitä enemmän yritys pystyy uudelleenkäyttämään aiemmin luotua ohjelmistoa, sitä vähemmän uusien tuotteiden kehitykseen kuluu aikaa ja rahaa. Ohjelmiston uudelleenkäyttöön kykenevä yritys saa siten tuotteensa markkinoille aikaisemmin ja edullisemmalla hinnalla kuin sen uudelleenkäyttöön kykenemättömät kilpailijat.

4.2 Ohjelmiston systemaattinen uudelleenkäyttö

Monet teknologiayritykset pyrkivät systemaattiseen ohjelmiston uudelleenkäyttöön. Systemaattisella uudelleenkäytöllä tarkoitetaan sovellusalakohtaisen ohjelmiston ja siihen liittyvien korkean tason artefaktien (vaatimusmäärittelyt, testitapaukset jne.) järjestelmällistä uudelleenkäyttöä koko organisaatiossa [26]. Systemaattinen ohjelmiston uudelleenkäyttö ei tapahdu itsestään, eikä yhtä yleispätevää ratkaisua sen toteuttamiseksi ole. W.B. Frakes ja S. Isoda esittävät systemaattista ohjelmiston uudelleenkäyttöä koskevassa artikkelissaan [26] joitakin yleispäteviä onnistuneen systemaattisen uudelleenkäytön tekijöitä. Näitä tekijöitä ovat muun muassa uudelleenkäyttöä tukeva johtaminen, sovellusala-analyysi ja uudelleenkäytettävän ohjelmiston helppo saatavuus.

Systemaattisen ohjelmiston uudelleenkäytön on oltava ylhäältä alas johdettua [26]. Ohjelmiston uudelleenkäyttöön liittyy riskejä, joten yksittäiset projektipäälliköt voivat epäröidä omien resurssiensa kohdentamista uudelleenkäytettävien komponenttien kehittämiseen. Uudelleenkäytettävyys on ylimääräinen vaatimus ohjelmistolle, ja sen toteuttaminen vaatii enemmän aikaa ja rahaa kuin ad-hoc-ratkaisu käsillä olevaan ongelmaan. Ohjelmiston uudelleenkäyttö voi epäonnistua, eli ohjelmistoa ei huonon suunnittelun vuoksi voidakaan käyttää uudelleen, tai uudelleenkäytettäväksi suunnitellulle ohjelmistolle ei yksinkertaisesti ole käyttöä tulevaisuudessa. Epäonnistuessaan uudelleenkäytön mahdollistamiseen käytetty aika ja raha on mennyt hukkaan. Onnistuessaankin ohjelmiston uudelleenkäytön takaisinmaksuaika voi olla useita vuosia. Yksittäiset projektipäälliköt tarvitsevat siksi ylemmän johdon tukea ja rahoitusta päätöksilleen luoda uudelleenkäytettäviä komponentteja.

J. Long käsittelee uudelleenkäytön antimalleja käsittelevässä artikkelissaan [27] virheitä, joita yritykset tyypillisesti tekevät pyrkiessään systemaattiseen uudelleenkäyttöön. Monet näistä virheistä liittyvät johtamiseen. Yrityksen johtajat eivät voi olettaa, että ohjelmistosta tulee uudelleenkäytettävää, jos he vain päättävät niin. Uudelleenkäytettävän ohjelmiston luominen on työlästä, joten yrityksen johdon on tarjottava kehittäjille riittävästi

aikaa ja rahoitusta uudelleenkäytettävän ohjelmiston toteuttamiseen omien projektiensa lisäksi. Hyvä ajatus on luoda erillinen kehitystiimi uudelleenkäytettäviä komponentteja varten, sillä markkinoille vietäviä lopputuotteita kehittävillä tiimeillä on aikataulujen vuoksi painetta lykätä tai sivuuttaa uudelleenkäytettävien komponenttien toteutus omassa kehitystyössään. Yleinen yritysjohdon harhaluulo on, että ohjelmiston uudelleenkäyttö tulee edullisemmaksi heti alusta lähtien. Tosiasiassa uudelleenkäyttöön liittyy kehityskustannuksia, ja voi mennä vuosia, ennen kuin uudelleenkäytettävä ohjelmisto maksaa itsensä takaisin. Systemaattinen ohjelmiston uudelleenkäyttö vaatii kärsivällisyyttä yrityksen johdolta.

Jotta systemaattinen ohjelmiston uudelleenkäyttö olisi mahdollisimman tehokasta, sovellusalueelta on tunnistettava yhteiset toiminnot ja ominaisuudet, jotka kannattaa toteuttaa uudelleenkäytettävinä ohjelmistokomponentteina. Uudelleenkäytön kohteiden etsimistä kutsutaan sovellusala-analyysiksi [26]. Sovellusala-analyysillä uudelleenkäytettävyyden parantamiseksi tehtävä työ saadaan kohdistettua oikeisiin asioihin. Sovellusala-analyysiin liittyy kuitenkin riski, jota Long kutsuu sovellusala-analyysiparalyysiksi [27]. Longin mukaan tyypillinen virhe, minkä yritykset tekevät, on odottaa sovellusala-analyysin valmistumista ennen kuin minkäänlaista systemaattista uudelleenkäyttöä aletaan toteuttamaan. Sovellusala-analyysi voi kestää kuukausia, ja ilman näkyviä tuloksia yrityksen johdolta voi loppua usko uudelleenkäytettävyysprojektin onnistumiseen.

Kun yritys on saanut toteutettua uudelleenkäytettäviä ohjelmistokomponentteja, ne on saatettava koko yrityksen saataville. Tyypillisesti yritykset perustavat uudelleenkäyttökirjaston, josta uudelleenkäytettävät komponentit ovat helposti löydettävissä ja saatavissa [26]. Longin mukaan yritykset voivat sortua uudelleenkäyttökirjaston luomisessa useisiin antimalleihin, mikä voi estää ohjelmiston uudelleenkäyttöä. Uudelleenkäyttökirjaston luominen ei itsessään takaa ohjelmiston uudelleenkäyttöä. Kirjaston olemassaoloa on mainostettava yrityksen sisällä, jotta uuden ohjelmiston kehittäjät ymmärtävät etsiä tarvitsemiaan komponentteja. Uudelleenkäyttökirjastosta on oltava olemassa ajantasainen katalogi, jotta kehittäjät löytävät etsimänsä. Uudelleenkäyttökirjastoon on sallittava vain riittävän geneerisiä ja hyvin testattuja komponentteja, jottei uudelleenkäytöstä koidu kehittäjille ylimääräistä työtä, ja jotta kehittäjät luottaisivat komponenttien laatuun. [27]

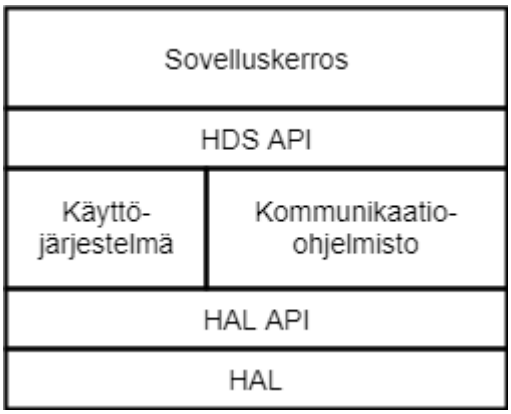
4.3 Laitteistoabstraktio

Koneenohjausjärjestelmän ohjelmisto on väistämättä riippuvainen käytössä olevasta laitteistosta. Ohjausyksikön ohjelmisto saa tietoa ympäristöstään ja ohjaa toimilaitteita luvussa 2.1.2 esitetyillä tavoilla. Ohjausyksiköiden IO-rajapinta, anturien mittaustulosten esitysmuoto ja toimilaitteiden vaatima ohjaus voivat vaihdella eri valmistajien välillä. Koneiden valmistajat saattavat toisinaan vaihtaa ohjausyksiköiden, anturien ja toimilaitteiden toimittajia, tai jokin toimittaja saattaa lopettaa toimintansa. Koneenohjausjärjestelmän ohjelmisto ei saa olla riippuvainen tietyn valmistajan antureista, toimilaitteista tai

ohjausyksiköistä, jotta toimittajan vaihtaminen olisi mahdollista ilman kohtuuttomia muutoksia koneenohjausjärjestelmään.

Koneenohjausjärjestelmissä yleinen tapa pienentää laitteistoriippuvuutta on laitteistoabstraktio (HAL, Hardware Abstraction Layer) [7, s. 264-269]. HAL tarkoittaa ohutta ohjelmistokerrosta, joka on täysin riippuvainen laitteistosta [25, s. 67-94]. HAL tarjoaa abstrahoidun rajapinnan laitteiston resursseihin, jotta sovelluslogiikan ei tarvitsisi olla suoraan riippuvainen laitekohtaisista yksityiskohdista. HAL on ohut, eli se ei sisällä sovelluskohtaista logiikkaa. Laitteiston vaihtamiseksi tarvitsee vaihtaa vain HAL:n toteutus, mikä on paljon helpompaa kuin etsiä ja korjata kaikki laitteistoviittaukset ilman HAL:aa toteutetusta sovelluslogiikasta. Kunhan uusi HAL toteuttaa saman rajapinnan kuin alkuperäinen, sovelluslogiikkaan ei tarvita mitään muutoksia, ja se toimii edelleen odotetulla tavalla.

Perinteisessä ohjelmistopinossa, joka on esitetty kuvassa 11, HAL on kaikkein alin ohjelmistokerros, joka on suoraan vuorovaikutuksessa fyysisen laitteiston kanssa. Seuraavaksi alimman kerroksen muodostavat käyttöjärjestelmä ja kommunikaatio-ohjelmisto. Käyttöjärjestelmä vastaa laitteen resurssien, kuten suoritusajan ja muistin, jakamisesta sovelluskerroksen sovelluksille ja resurssien käytön valvonnasta. Kommunikaatio-ohjelmisto tarjoaa sovelluksille mahdollisuuden kommunikoida keskenään tai ulkoisten laitteiden kanssa. Ylin kerros on sovelluskerros, joka koostuu järjestelmän varsinaisen toimintalogiikan toteuttavasta ohjelmistosta. Riippuen käyttöjärjestelmästä, sovelluskerros voi koostua yhdestä tai useammasta rinnakkain suoritettavasta sovelluksesta. Sovellukset eivät käytä suoraan HAL:n rajapintaa (HAL API), vaan käyttöjärjestelmän ja kommunikaatio-ohjelmiston tarjoamaa laitteistoriippuvan ohjelmiston rajapintaa (HDS API, Hardware Dependent Software Application Programming Interface). Käyttöjärjestelmä ja kommunikaatio-ohjelmisto käyttävät omassa toteutuksessaan HAL:n rajapintaa laitteiston resurssien käyttämiseksi. [25, s. 67-94]



Kuva 11. *Perinteinen ohjelmistopino [25, s. 67-94].*

Perinteisessä ohjelmistopinossa HAL abstrahoi sitä laitteistoa, jolla sovelluksia ajetaan, eli ohjausyksikköä. Koneenohjausjärjestelmissä tarvitaan lisäksi laitteistoriippuvainen

abstraktiokerros abstrahoimaan laajemmin koko ohjattavaa järjestelmää: ohjausyksikköön liitettyjä antureita ja toimilaitteita. Perinteisessä ohjelmistopinossa ulkoisia laitteita abstrahoiva ohjelmistokomponentti sijoittuu sovelluskerrokseen, koska ohjausyksikön oma HAL ei voi sisältää sovellusriippuvaisia ominaisuuksia. Ulkoisten laitteiden HAL on kuitenkin samaan tapaan ohut ja täysin laitteistoriippuvainen, kuten ohjausyksikön oma HAL. Ulkoisten laitteiden HAL:n keskeisenä tehtävänä on muuntaa anturien raaka-arvot sovelluslogiikan kannalta mielekkääseen muotoon, esimerkiksi SI-yksiköiksi. Vastaavasti HAL muuntaa sovelluslogiikan tuottamat ohjauskäskyt toimilaitteiden vaatimaan muotoon, esimerkiksi prosentteina ilmoitetun jarrutuspyynnin jarruventtiilin ohjausvirraksi.

Ulkoisten laitteiden HAL on toteutettava joustavaksi siten, että ulkoisia antureita ja toimilaitteita voidaan vaihtaa koneen koko elinkaaren aikana ilman ohjelmistopäivityksiä. Koneisiin tarvitsee yleensä niiden elinkaaren aikana asentaa varaosia, eikä alkuperäisiä osia välttämättä ole enää saatavilla. Korvaavat osat saattavat toimia hieman eri tavalla kuin alkuperäiset. Esimerkiksi uuden lämpötila-anturin arvoalue voi olla erilainen, tai uusi jarruventtiili voi tarvita vähemmän virtaa kuin alkuperäinen. Antureissa ja toimilaitteissa voi olla myös pieniä yksilöllisiä eroja, ja niiden ominaisuudet voivat muuttua laitteen vanhetessa ja kuluessa. Anturien ja toimilaitteiden on siksi oltava koneen käyttäjän kalibroitavissa.

Ratkaisuna sekä osien vaihtamiseen että kalibrointiin on konfiguroitava HAL. Anturien mittaustulosten muunnoskertoimet, toimilaitteiden ohjauksen raja-arvot ja muut muuttuvat ominaisuudet tallennetaan ohjausyksikön pysyvään muistiin. Pysyvään muistiin tallennettuja tietoja päivitetään varaosan asennuksen tai kalibroinnin yhteydessä. Edistyneet koneenohjausjärjestelmät tarjoavat ohjatut toiminnot koneen osien kalibrointiin, eikä koneen käyttäjän tarvitse välittää siitä, mitä tietoja kalibrointi vaatii, ja miten ne on tallennettu ohjausyksikön muistiin.

4.4 Väyläabstraktio

Luvussa 2.3 esiteltiin eri tapoja, joilla ohjausyksiköt, IO-moduulit sekä älykkäät anturit ja toimilaitteet voivat vaihtaa tietoa keskenään kommunikaatioväylillä. Luvussa kerrottiin, että vaihtoehtoisia kommunikaatioväylätoteutuksia on useita, ja kullakin väylätoteutuksella voi olla useita vaihtoehtoisia kommunikaatioprotokollia. Ohjausyksiköllä voi olla käytössä useita väyliä, eivätkä ne välttämättä ole kaikki samanlaisia. Väylätoteutusten ja protokollien monimuotoisuus vaikeuttaa ohjelmiston uudelleenkäyttöä. Ohjatakseen esimerkiksi IO-moduuliin kytkettyä toimilaitetta ohjausyksikön on lähetettävä ohjauskäsky oikealle väylälle, oikealla protokollalla, oikealla baudinopeudella, oikealle vastaanottajalle ja kirjoitettava viesti oikeaan dataobjektiin (CANopen), parametriryhmään (J1939) tai määritellä ohjattava toiminto mulla protokollan vaatimalla tavalla. Jos mikä tahansa näistä yksityiskohdista muuttuu koneenohjausjärjestelmän kehitystyön aikana tai eri tuotteiden välillä, ohjausjärjestelmän toteutukseen on tehtävä muutoksia. On varsin

yleistä, että väylärakenne muuttuu toistuvasti koneen kehitysvaiheessa, ja kahdella eri mallisella koneella ei ole käytännössä koskaan täysin samanlaista väylärakennetta.

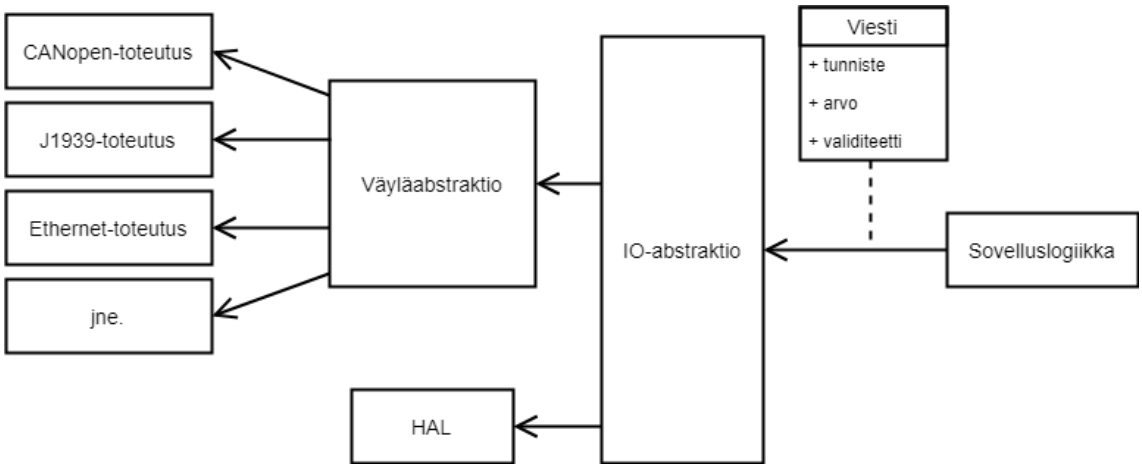
Väyläabstraktio on hajautetuissa koneenohjausjärjestelmissä yleisesti käytetty tapa välttää ohjelmistomuutoksia järjestelmän väylärakenteen muuttuessa [7, s. 143]. Väyläabstraktiossa järjestelmän väylärakenne piilotetaan sovelluslogiikalta ohjelmistorajapinnan taakse. Sovelluslogiikka lähettää ja vastaanottaa väyläviestit rajapinnan kautta, eikä siten ole suoraan riippuvainen väylien toteutusyksityiskohdista. Jos jokin asia väylärakenteessa muuttuu, muutokset koskevat vain väyläabstraktion toteuttavaa ohjelmistoa. Hyvin suunnitellussa väyläabstraktiossa kaikki väylärakenteen yksityiskohdat ovat säädettävissä konfiguraatiolla, eikä ohjelmistomuutoksia tarvita ollenkaan.

Koneenohjauksen turvallisuuden parantamiseksi väyläabstraktion on osattava ilmoittaa, jos väylälle lähetettävät viestit eivät pääsekään perille tai vastaanotettu viesti on päässyt vanhentumaan esimerkiksi katkenneen väylän vuoksi. Koneenohjausjärjestelmä pystyy silloin huomioimaan vikatilanteet, ja asettumaan turvalliseen tilaan, kunnes vika on korjaantunut. Kuhunkin väyläviestiin liitetty validiteettitieto on yleinen tapa ilmoittaa vikatilanteista sovelluslogiikalle [7, s. 330-335].

Väyläabstraktion rajapinta on suunniteltava riittävän geneerisiksi siten, että se on mahdollista toteuttaa kaikilla mahdollisilla väylätoteutuksilla ja -protokollilla. Rajapinnan on oltava yksinkertainen, eikä rajapinnan käyttäjän pidä olla tietoinen väylien lukumääristä, niiden tyypeistä ja protokollista tai laitteiden sijoittelusta eri väylille. Yksinkertaisimmillaan rajapinta voi koostua viestialkioista, joiden ominaisuuksia ovat tunniste, arvo ja validiteetti. Viestialkio kuvaa yksittäistä lähetettävää tai vastaanotettavaa viestiluokkaa, esimerkiksi lämpötila-anturin mittaustietoa tai jarruventtiilin ohjausta. Viestialkion tunniste on yksilöllinen, ja sovelluslogiikka viittaa viestiin tunnistetta käyttäen. Väyläabstraktio osaa tunnisteen avulla etsiä konfiguraatiostaan oikean väylän, vastaanottajan ja dataobjektin, jolle viesti on lähetettävä, tai mistä se on vastaanotettava. Tunniste ei saa olla riippuvainen väylärakenteesta, vaan sen on pysyttävä samana, jos esimerkiksi laitteita sijoitellaan uudelleen eri väylille. Viestialkion arvo on viimeisin lähetetty tai vastaanotettu arvo. Viestialkion validiteetti ilmoittaa mahdollisista vikatilanteista viestin lähetyksessä tai vastaanotossa.

Väyläviestien ja ohjausyksikön omien fyysisten IO-liitäntöjen lukeminen ja kirjoittaminen kannattaa abstrahoida yhteisen ohjelmistorajapinnan taakse. Tällä tavalla sovelluslogiikka voi olla täysin riippumaton toimilaitteiden ja anturien sijoittelusta. Ohjausyksikön fyysisiin IO-liitäntöihin kytketyt anturit ja toimilaitteet on mahdollista siirtää IO-moduulin ohjattavaksi tai korvata esimerkiksi suoraan CAN-väylää käyttävillä älykkäillä laitteilla ilman ohjelmistomuutoksia. Yhteinen abstraktiokerros käyttää fyysisten IO-liitäntöjen lukemiseen ja kirjoittamiseen HAL-rajapintaa, ja väyläviestien lukemiseen ja kir-

joittamiseen väyläabstraktion rajapintaa. Yleisen IO-abstraktion rajapinta voi koostua samanlaisista viestialkioista kuin väyläabstraktion rajapintakin. Yleisen IO-abstraktion periaate on esitetty kuvassa 12.



Kuva 12. *IO-abstraktio yhtenäistää väyläviestien ja muun IO:n käsittelyn.*

4.5 Olio-ohjelmoinnin SOLID-periaatteet

Kuten luvussa 3.3.2 todettiin, olio-ohjelmointi soveltuu hyvin uudelleenkäytettävän ohjelmiston toteutusparadigmaksi. Valitusta olio-ohjelmointikielestä riippumatta olio-ohjelmoinnissa voidaan soveltaa tiettyjä hyväksi todettuja periaatteita, joita kutsutaan SOLID-periaatteiksi. SOLID on akronyymi englannin kielen sanoista:

- S = Single responsibility principle, yhden vastuualueen periaate
- O = Open/Closed principle, avoin/suljettu -periaate
- L = Liskov substitution principle, Liskovin substituutioperiaate
- I = Interface segregation principle, rajapintojen eriyttämisen periaate
- D = Dependency inversion principle, riippuvuuksien kääntämisen periaate. [28]

SOLID-periaatteet eivät liity vain uudelleenkäytettävän ohjelmiston suunnitteluun, vaan yleisesti hyvään ohjelmistosuunnitteluun. Kukin periaate kuitenkin tukee uudelleenkäytettävyyttä omalla tavallaan. Periaatteet on selitetty yksityiskohtaisemmin seuraavissa aliluvuissa.

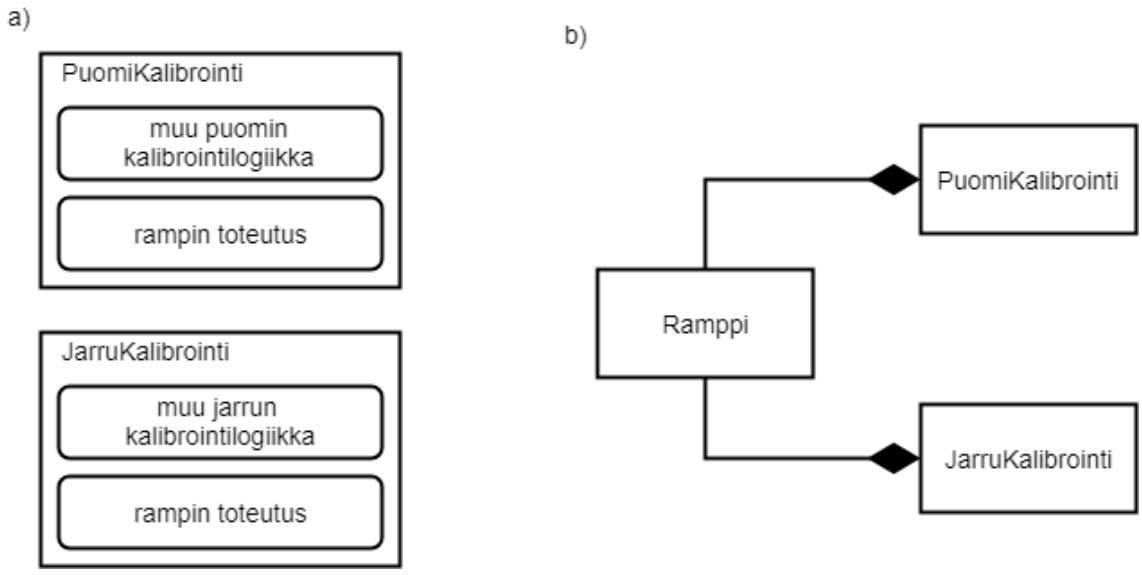
4.5.1 Yhden vastuualueen periaate

Yhden vastuualueen periaatteen mukaan kullakin luokalla, moduulilla tai muulla vastaavalla ohjelmointikielen tarjoamalla rakenteella tulee olla vain yksi vastuualue. Luokkaa tarvitsee muokata jälkikäteen vain, jos sen ainoan vastuualueen vaatimukset muuttuvat. [23, s. 133-166][28][29] Periaatteen noudattaminen johtaa hienojakoiseen luokkajakoon,

mikä hyödyttää kyseisten luokkien uudelleenkäyttöä. Hienojakoisia luokkia yhdistelemällä voidaan luoda uusia luokkia, jotka käyttävät uudelleenkäytettäviä luokkia eri tarkoitukseen, kuin mitä varten ne oli alun perin toteutettu.

Yhden vastuualueen periaatetta rikkova luokkajako vastaavasti vaikeuttaa uudelleenkäyttöä, tai tekee siitä mahdotonta. Huonosti suunnitellussa luokkajaossa vastuualueiden toteutukset on tiukasti sidottu toisiinsa, eikä yhden vastuualueen toteutusta voida irrottaa käytettäväksi uudelleen toisaalla. Tiukka sidonta johtaa helposti saman koodin kopioimiseen useaan eri paikkaan, mikä vaikeuttaa ohjelmiston ylläpitoa.

Kuva 13 havainnollistaa yhden vastuualueen periaatteen merkitystä ohjelmiston uudelleenkäytön kannalta. Kuvan esimerkissä kaksi eri luokkaa (PuomiKalibrointi ja JarruKalibrointi) tarvitsevat ramppifunktiota omassa toteutuksessaan. Kuvassa a) ei ole noudatettu yhden vastuualueen periaatetta, minkä vuoksi ramppifunktio on täytynyt toteuttaa uudelleen molemmissa luokissa. Kuvassa b) kahdennettu toteutus on vältetty toteuttamalla ramppifunktio yhden vastuualueen periaatteen mukaisesti omassa luokassaan (Ramppi), jota on käytetty sekä puomin että jarrun kalibroinnin toteutuksessa.



Kuva 13. *Esimerkki yhden vastuualueen periaatteen soveltamisesta.*

Yhden vastuualueen periaate pienentää myös regression eli jo kerran toimivaksi todetun ominaisuuden rikkoutumisen riskiä. Jos luokalla on vain yksi vastuualue, ja vastuualueen toteutusta täytyy muuttaa, muutoksen ei pitäisi vaikuttaa muiden vastuualueiden toteutuksen oikeellisuuteen, koska ne on toteutettu eri luokissa. Jos luokalla sen sijaan on useita vastuualueita, yhden vastuualueen toteutusta muuttaessaan ohjelmoija voi vahingossa tulla muuttaneeksi myös muiden vastuualueiden toteutusta. Regression riski kasvattaa kattavan testaamisen tarvetta pientenkin muutosten tapauksessa, mikä tekee ohjel-

miston ylläpidosta ja jatkokehittämisestä työlästä. Yhden vastuualueen periaatetta noudattavan luokan yksikkötestaaminen on myös helpompaa. Yksikkötestaus on usein automatisoitu, joten muutokset vaativat vain vähän manuaalista testaustyötä.

4.5.2 Avoin/suljettu -periaate

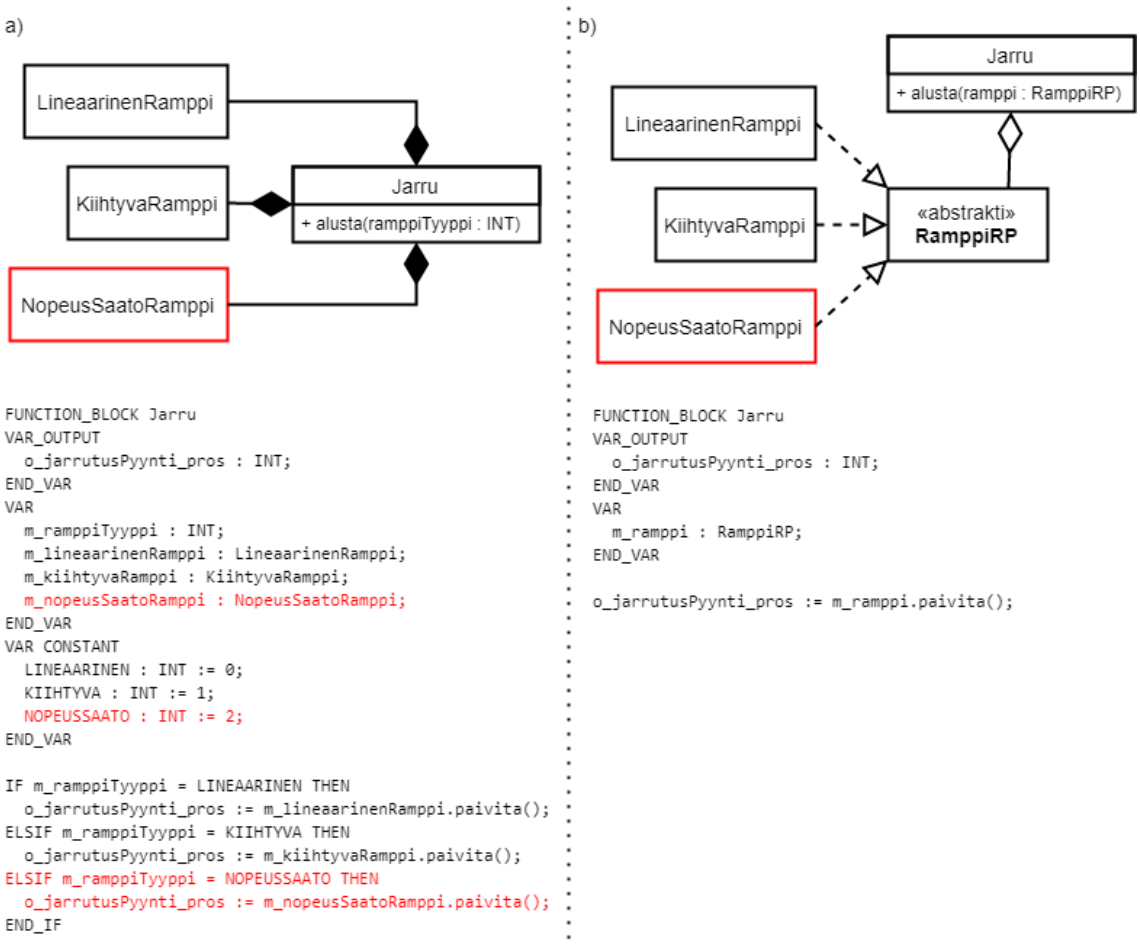
Avoin/suljettu -periaatteen mukaan ohjelmiston tulee olla avoin laajentamiselle, mutta suljettu muutoksilta [23, s. 133-166][28][29]. Käytännössä periaate tarkoittaa, että uusien ominaisuuksien lisäämisen ohjelmistoon pitää onnistua ilman muutoksia olemassa oleviin komponentteihin. Periaatetta noudattavassa ohjelmistossa uusien ominaisuuksien lisääminen on helppoa, koska uuden ominaisuuden kehittäjän tarvitsee tietää vain vähän olemassa olevasta ohjelmistosta toteuttaakseen uuden ominaisuuden. Uuden ominaisuuden lisäämisestä aiheutuva regression riski on pieni, koska olemassa olevaan, jo kertaalleen toimivaksi todettuun ohjelmistoon ei tehdä muutoksia.

Periaatteen noudattamisessa olio-ohjelmoinnin periytymismekanismilla, ja etenkin abstrakteilla rajapinnoilla on keskeinen merkitys. Ohjelmiston laajentaminen uusilla ominaisuuksilla ilman muutoksia olemassa oleviin komponentteihin ei ole mahdollista, jos komponentit ovat tiukasti sidottu toisiinsa. Ohjelmisto on toteutettava siten, että vaihdettavien tai laajennettavien komponenttien käyttäjät viittaavat niihin vain abstraktin rajapinnan tai yhteisen kantaluokan kautta.

Kuvassa 14 on esimerkki avoin/suljettu -periaatteen soveltamisesta. Koneen jarrutuksen alkuperäisessä toteutuksessa jarrutus oli mahdollista kahdella tavalla: lineaarisella rampilla ja kiihtyvällä rampilla. Nyt koneeseen halutaan lisätä uusi jarrutusramppi, jossa jarrua säädetään koneen nopeuden mukaan. Rampit on toteutettu yhden vastuualueen periaatteen mukaisesti omissa luokissaan. Kuvassa a) jarrun toteutus on suoraan riippuvainen kustakin vaihtoehtoisesta rampin toteutuksesta, ja valitsee käytettävän toteutuksen alustuksessa annetun parametrin mukaan. Kuvan a) jarrutuksen toteutus ei noudata avoin/suljettu -periaatetta. Kolmannen ramppitoteutuksen lisääminen pakottaa tekemään muutoksia olemassa olevaan jarrutuksen toteutukseen useaan eri kohtaan. Muutokset on merkitty kuvassa punaisella. Esimerkissä vaadittavat muutokset eivät ole isoja, mutta jos kehittäjä on ajan kuluessa unohtanut alkuperäisen toteutuksen yksityiskohdat, tai jos muutoksen tekijä on eri kuin alkuperäinen kehittäjä, häneltä kuluu ylimäärästä aikaa löytää ne kohdat ohjelmistosta, jonne muutokset on tehtävä. Kun uuden rampin lisäys on tehty, kaikki vaihtoehtoiset jarrutustavat on testattava huolellisesti uudelleen, koska kehittäjä on voinut koodia muuttaessaan tehdä virheen, joka vaikuttaa vanhoihin jarrutustapoihin.

Kuvassa b) on noudatettu avoin/suljettu -periaatetta. Vaihtoehtoisilla ramppitoteutuksilla on yhteinen rajapinta, ja jarrun toteutus on riippuvainen vain tästä rajapinnasta. Jarrulle annetaan alustuksessa viite oikeaan ramppitoteutukseen. Ainoat muutokset ohjelmistoon ovat uuden ramppitoteutuksen lisäämien ja oikean toteutuksen valitseminen jarrua alustettaessa. Jälkimmäinen saattaa vaatia muutosta olemassa olevaan koodiin, mutta sitä ei

voida välttää kummassakaan jarrun toteutustavassa. Uuden vaihtoehtoisen rampin lisääminen ei vaadi lainkaan muutoksia jarrun toteutukseen. Kehittäjän ei tarvitse tietää jarrun toteutusyksityiskohtia lisätäkseen uuden rampin. Vanhat jarrutustavat eivät ole voineet muuttua uuden rampin lisäämisen seurauksena, mikä pienentää regressiotestauksen tarvetta.




```
FUNCTION_BLOCK Jarru
VAR_OUTPUT
  o_jarrutusPyynti_pros : INT;
END_VAR
VAR
  m_ramppiTyyppi : INT;
  m_lineaarinenRamppi : LineaarinenRamppi;
  m_kiihtyvaRamppi : KiihtyvaRamppi;
  m_nopeusSaatoRamppi : NopeusSaatoRamppi;
END_VAR
VAR CONSTANT
  LINEAARINEN : INT := 0;
  KIIHTYVA : INT := 1;
  NOPEUSSAATO : INT := 2;
END_VAR

IF m_ramppiTyyppi = LINEAARINEN THEN
  o_jarrutusPyynti_pros := m_lineaarinenRamppi.paivita();
ELSIF m_ramppiTyyppi = KIIHTYVA THEN
  o_jarrutusPyynti_pros := m_kiihtyvaRamppi.paivita();
ELSIF m_ramppiTyyppi = NOPEUSSAATO THEN
  o_jarrutusPyynti_pros := m_nopeusSaatoRamppi.paivita();
END_IF
```

```
FUNCTION_BLOCK Jarru
VAR_OUTPUT
  o_jarrutusPyynti_pros : INT;
END_VAR
VAR
  m_ramppi : RamppiRP;
END_VAR

o_jarrutusPyynti_pros := m_ramppi.paivita();
```

Kuva 14. *Esimerkki avoin/suljettu -periaatteen soveltamisesta.*

4.5.3 Liskovin substituutioperiaate

Liskovin substituutioperiaate vaatii, että minkä tahansa luokan instanssi tai rajapinnan toteutus voidaan korvata aliluokan instanssilla tai rajapinnan vaihtoehtoisella toteutuksella vaikuttamatta ohjelman oikeellisuuteen [23, s. 133-166][28][29]. Periaate tunnetaan myös nimellä is-a -periaate: Jokainen aliluokan instanssi on samalla myös kantaluokkansa instanssi, ja kantaluokan käyttäjän näkökulmasta instanssin todellisella tyypillä ei ole merkitystä.

Liskovin substituutioperiaatteen noudattamiseen liittyy läheisesti niin kutsuttu sopimussuunnittelu. Sopimussuunnittelussa määritellään selkeästi rajapinnan kutsujan ja toteuttajan vastuut. Kutsujan vastuita kutsutaan esiehdoiksi. Esiehtoja voivat olla esimerkiksi rajapinnan metodeille annettavien parametrien arvoalueet tai metodien kutsumajärjestys. Rajapinnan toteuttajan vastuita kutsutaan jälkiehdoiksi. Jälkiehdot kuvaavat, mitä rajapinnan palvelut tekevät, olettaen että palvelun kutsuja on täyttänyt esiehdot. Rajapinnan esi- ja jälkiehdot on dokumentoitava selkeästi, vähintään koodikommentteina rajapinnan määrittelyssä. Hyvin dokumentoitujen esi- ja jälkiehtojen ansiosta rajapinnan käyttäjän ei tarvitse tutustua rajapinnan toteutukseen tietääkseen, kuinka rajapintaa kuuluu käyttää.

Liskovin substituutioperiaatteen mukaan rajapinnan esi- ja jälkiehtojen on pysyttävä voimassa kaikissa rajapinnan vaihtoehtoisissa toteutuksissa ja periytetyissä aliluokissa. Rajapinnan uudet toteutukset eivät voi vaatia tiukempia esiehtoja, ja niiden on toteutettava vähintään rajapinnan dokumentaatiossa määritellyt jälkiehdot [23, s. 133-166] Muussa tapauksessa rajapinnan käyttäjän pitäisi huomioida uuden toteutuksen poikkeavat ehdot omassa koodissaan, mikä helposti johtaisi avoin/suljettu -periaatteen rikkomiseen. Liskovin substituutioperiaate liittyy siten ohjelmiston uudelleenkäyttöön avoin/suljettu -periaatteen mahdollistavana tekijänä.

Riittävä dokumentaatio on ohjelmiston uudelleenkäytön kannalta tärkeää. Mitä enemmän kehittäjä joutuu näkemään vaivaa ymmärtääkseen rajapinnan käyttöä, sitä todennäköisemmin hän hylkää uudelleenkäytön ja toteuttaa tarvitsemansa palvelut itse uudelleen. Huono tai puuttuva dokumentaatio voi johtaa rajapinnan väärinkäytöstä aiheutuviin bugeihin, joiden ratkomiseen kuluu aikaa ja vaivaa, eikä kannusta ohjelmiston uudelleenkäyttöön jatkossa. Sopimussuunnittelun mukainen esi- ja jälkiehtojen määrittely tarjoaa hyvän mallin rajapinnan dokumentaatiolle.

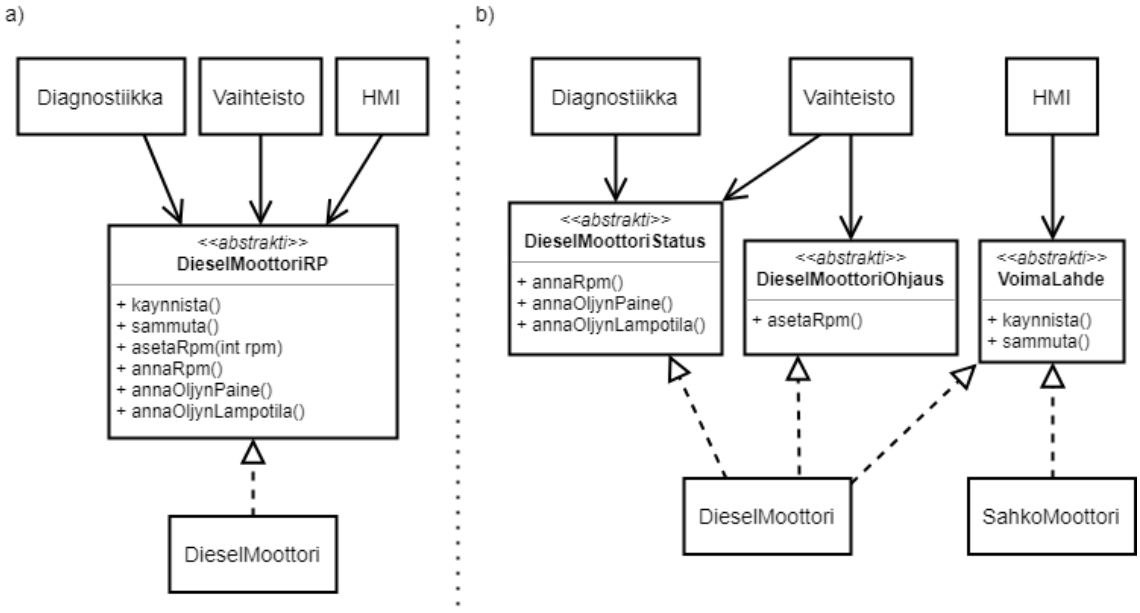
4.5.4 Rajapintojen eriyttämisen periaate

Rajapintojen eriyttämisen periaatteen mukaan abstraktien rajapintojen suunnittelussa kannattaa suosia pieniä käyttötapauskohtaisia rajapintoja suurten monoliittisten rajapintojen sijaan [23, s. 133-166][28]. Suuret kokonaisuudet voivat toteuttaa useita pieniä rajapintoja, ja suuria rajapintoja on mahdollista koostaa pienistä rajapinnoista useiden olio-ohjelmointikielten tarjoaman rajapintojen laajentamisen avulla. Rajapintojen eriyttämisen periaatteella saavutetaan kaksi hyötyä: rajapintojen helppokäyttöisyys ja mahdollisuus toteuttaa uudelleen vain osa monoliittisesta rajapinnasta.

Eriytettyjen rajapintojen käyttäminen on helpompaa kuin monoliittisten rajapintojen käyttäminen. Eriytetyt rajapinnat tarjoavat käyttäjälle vain minimaalisen määrän tietyn käyttötapauksen toteuttamiseen tarvittavia palveluita. Rajapinnan käyttäjällä on siten vähemmän opeteltavaa, ja rajapinnan väärinkäyttömahdollisuudet on minimoitu. [28]

Eriytettyjä rajapintoja on mahdollista toteuttaa uudelleen toteuttamatta kaikkia monoliittisen rajapinnan palveluita. Tämä ehkäisee rajapintojen vaihtoehtoisten toteutusten yhteisen logiikan toistamista. Toisinaan uudessa toteutuksessa kaikille lopuille monoliittisen rajapinnan palveluille ei ole mahdollista tarjota mielekästä toteutusta, mikä voi johtaa Liskovin substituutioperiaatetta rikkoviin tynkätoteutuksiin. [23, s. 133-166]

Kuvassa 15 on esimerkki rajapintojen eriyttämisen periaatteen soveltamisesta. Dieselmoottori on monimutkainen komponentti, joka tarjoaa moninaisia moottoriin liittyviä palveluita. Moottorilla on useita eri käyttäjiä, jotka ovat kiinnostuneita eri palveluista. Koneen diagnostiikka tarvitsee tietoa moottorin hetkellisestä kierrosnopeudesta, öljyn paineesta, lämpötilasta ja niin edelleen. Vaihteisto tarvitsee tietoa moottorin kierrosnopeudesta, ja sen on lisäksi pystyttävä säätämään sitä. HMI (Human-Machine Interface, koneen käyttöliittymä) käynnistää tai sammuttaa moottorin operaattorin pyynnöstä.



Kuva 15. *Esimerkki rajapintojen eriyttämisen periaatteen soveltamisesta.*

Kuvassa 15 a) dieselmoottori toteuttaa yhden monoliittisen rajapinnan, joka sisältää kaikki dieselmoottorin tarjoamat palvelut. Ratkaisu ei noudata rajapintojen eriyttämisen periaatetta. Kaikki rajapinnan käyttäjät suoriutuvat omista tehtävistään rajapinnan avulla, mutta rajapinta tarjoaa niille myös paljon tarpeettomia palveluita. Esimerkiksi diagnostiikan ei tarvitse pystyä käynnistämään moottoria, eikä sen missään nimessä kuulu tehdä niin. Monoliittinen rajapinta kuitenkin mahdollistaa moottorin käynnistymisen diagnostiikan ohjelmointivirheen seurauksena.

Toinen ongelma kuvan 15 a) ratkaisussa on HMI-moduulin uudelleenkäyttö koneessa, jossa onkin dieselmoottorin sijaan sähkömoottori. HMI vain käynnistää tai sammuttaa moottorin, mitkä ovat yhteisiä palveluita sekä diesel- että sähkömoottoreille. Kuvan a) ratkaisussa HMI on kuitenkin riippuvainen dieselmoottorin rajapinnasta. Jos HMI-toteu-

tusta haluttaisiin käyttää uudelleen avoin/suljettu -periaatteen mukaisesti ilman muutoksia sähkökoneessa, sähkömoottorin pitäisi toteuttaa dieselmoottorin rajapinta. Sähkömoottori ei kuitenkaan pysty tarjoamaan mielekästä vastinetta esimerkiksi kierrosnopeuden asettamiselle, joten rajapinnan toteutus rikkoisi Liskovin substituutioperiaatetta.

Kuvassa 15 b) monoliittinen rajapinta on eriytetty pienempiin roolikohtaisiin rajapintoihin. DieselMoottoriStatus-rajapinta tarjoaa diagnostiikan tarvitseman rajapinnan moottorin diagnostiikkatietojen lukemiseen ilman mahdollisuutta ohjata moottoria. DieselMoottoriOhjaus-rajapinta tarjoaa vaihteiston tarvitseman mahdollisuuden säätää moottorin kierrosnopeutta. VoimaLahde-rajapinta tarjoaa HMI:lle moottorin tyypistä riippumattoman rajapinnan moottorin käynnistämiseen ja sammuttamiseen.

Kuvassa 15 b) rajapintojen eriyttämisen periaate on ratkaissut molemmat kuvassa a) esiintyneet ongelmat. Diagnostiikalla on käytössään vain luku -rajapinta, eikä se voi väärin toimiessaankaan vaikuttaa moottorin toimintaan, mikä parantaa koneen turvallisuutta. HMI ei ole enää riippuvainen dieselmoottorista, vaan kaikille moottorityypeille yhteisestä rajapinnasta. HMI-moduulia voidaan siten käyttää uudelleen sähkökoneessa ilman muutoksia. Sähkömoottorin tarvitsee vain toteuttaa yhteinen VoimaLahde-rajapinta, minkä pitäisi onnistua rikkomatta Liskovin substituutioperiaatetta.

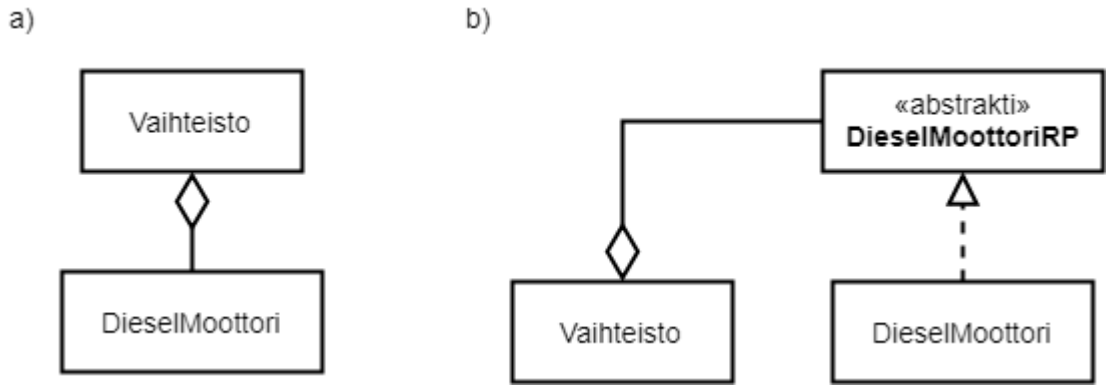
4.5.5 Riippuvuuksien kääntämisen periaate

Riippuvuuksien kääntämisen periaate poistaa suorat riippuvuussuhteet konkreettisten ohjelmistomoduulien välillä abstraktioiden avulla. Robert C. Martin muotoili riippuvuuksien kääntämisen periaatteen kahdella lauseella:

1. Korkean tason moduulit eivät riipu matalan tason moduuleista, vaan molemmat riippuvat abstraktioista.
2. Abstraktiot eivät riipu yksityiskohdista, vaan yksityiskohdat riippuvat abstraktioista. [23, s. 133-166]

Käytännössä periaate tarkoittaa, että ohjelmistomoduulien väliset riippuvuudet toteutetaan epäsuorasti abstraktien rajapintojen tai muun ohjelmointikielen tarjoaman abstraktiomekanismin avulla. Tällä tavalla moduuleja on mahdollista käyttää uudelleen erillään alkuperäisistä riippuvuuksistaan. Kullekin uudelleenkäytettävän moduulin tarvitsemalle rajapinnalle voidaan tarvittaessa tehdä uusi toteutus, mikäli alkuperäinen toteutus ei sovellu uuteen käyttötarkoitukseen. Liskovin substituutioperiaatetta noudattamalla uudelleenkäytettävä moduuli toimii oikein rajapintojen toteutuksen vaihtamisesta huolimatta. Riippuvuuksien kääntämisen periaate ehkäisee myös moduulien itsenäistä uudelleenkäyttöä estävät, ja muutenkin huonosta suunnittelusta kielivät sykliset riippuvuussuhteet [23, s. 133-166]. Riippuvuuksien kääntämisen periaate on siten keskeinen ohjelmiston uudelleenkäytön kannalta.

Kuvassa 16 on esimerkki riippuvuuksien kääntämisen periaatteen soveltamisesta. Vaihteiston ohjauksen on huomioitava moottorin turbiinin kierrosnopeus vaihteita vaihtaessaan. Pienemmälle vaihteelle vaihtaminen moottorin kierrosnopeuden ollessa jo valmiiksi korkea voi vahingoittaa moottoria ja vaihdelaatikkoa. Kun operaattori pyytää pienempää vaihdetta, vaihteiston on tarvittaessa hidastettava moottoria ennen vaihteen vaihtamista. Vaihteisto on siten riippuvainen moottorista. Kuvassa a) vaihteiston toteutus on suoraan riippuvainen moottorin toteutuksesta, mikä rikkoo riippuvuuksien kääntämisen periaatetta. Ratkaisu haittaa vaihteiston toteutuksen uudelleenkäyttöä uudessa konemallissa. Vaihteisto on tiukasti riippuvainen tietystä moottorin toteutuksesta, ja jos uudessa konemallissa halutaan käyttää eri moottorimallia, vaihteistoon on tehtävä muutoksia, mikä rikkoo avoin/suljettu -periaatetta.



Kuva 16. *Esimerkki riippuvuuksien kääntämisen periaatteen soveltamisesta.*

Kuvassa 16 b) vaihteiston riippuvuus moottorin toteutuksesta on käännetty: Vaihteisto on riippuvainen korkeamman tason abstraktista rajapinnasta matalamman tason konkreettisen moduulin sijaan. Nyt vaihteiston toteutusta on mahdollista käyttää sellaisenaan uudessa konemallissa, vaikka siinä olisi erimallinen moottori kuin alkuperäisessä järjestelmässä. Tämä tietysti edellyttää, että uusi moottori toteuttaa abstraktin rajapinnan.

5. CASE: LASTAUSKONE

Tässä luvussa on esitelty Sandvikin uudelleenkäytettävää lastaus- ja kuljetuskoneiden koneenohjausjärjestelmää esimerkkinä siitä, miten aiemmissa luvuissa esitettyjä suunnittelumalleja ja periaatteita voidaan soveltaa käytännön kehitystyössä.

5.1 Projektin kuvaus ja tavoitteet

Sandvikin tavoitteena on luoda uusi koneenohjausjärjestelmä, joka soveltuu Sandvikin jatkossa kehittämille kaivoslastaus- ja -kuljetuskoneille. Jotta uutta koneenohjausjärjestelmää voidaan testata käytännössä, projektissa kehitetään myös ensimmäinen uuteen koneenohjausjärjestelmään perustuva prototyyppilastauskone. Projektissa kehitettävä ohjelmisto koostuu siten kahdesta osasta: konemallien välillä uudelleenkäytettävästä osasta ja kehitettävälle prototyyppikoneelle erityisestä osasta. Kutsuttakoon uudelleenkäytettävää osaa tässä diplomityössä nimellä UKKO (UudelleenKäytettävä KoneenOhjausjärjestelmä). UKKO:oon perustuvaan prototyyppikoneeseen ja sen ohjelmistoon viitataan jatkossa nimellä ’prototyyppi’.

UKKO ei ole ensimmäinen Sandvikin kehittämä uudelleenkäytettävä koneenohjausjärjestelmä lastaus- ja kuljetuskoneille. Sen edeltäjä kehitettiin jo 2000-luvun vaihteessa, ja se on edelleen käytössä ympäri maailmaa myytävissä lastaus- ja kuljetuskoneissa. Sitä ennen Sandvikin koneet oli toteutettu perinteisellä releohjauksella. UKKO:n edeltäjä on ollut parikymmentä vuotta käytössä, ja on suoriutunut ansiokkaasti vaatimuksistaan. Ajan kuluessa koneenohjausjärjestelmien vaatimukset ovat kuitenkin merkittävästi muuttuneet, eikä vanhan, osittain vanhentuneilla tekniikoilla toteutetun järjestelmän sovittaminen uusiin vaatimuksiin ollut järkevää.

Vanhan järjestelmän sähköistys kaipasi modernisointia ja yksinkertaistusta huollon ja kokoonpanon helpottamiseksi. Nykyaikaisissa ohjausyksiköissä on paljon enemmän muistia ja suorituskykyä kuin vanhan järjestelmän käyttämissä ohjausyksiköissä. Muistia ja parempaa suorituskykyä tarvitaan koneiden uusien monimutkaisten ominaisuuksien toteuttamiseksi. Nykyaikaiset kosketusnäyttötietokoneet mahdollistavat myös aiempaa paremman käyttäjäkokemuksen.

Uusia vaatimuksia, jotka UKKO toteuttaa, ovat digitalisaation vaatima tiedonkeräys ja uudet turvaominaisuudet. Digitalisaation aikana koneen käytöstä halutaan saada mahdollisimman paljon tietoa, jota voidaan käyttää tuottavuuden ja käyttäjäkokemuksen parantamiseen, tuotekehitykseen ja markkinointiin. Sandvik on kehittänyt tiedonkeruujärjestelmän, joka kerää tietoa CANopen-protokollaa käyttäviltä väyliltä [8] ja se on mahdollista asentaa myös UKKO:n edeltäjään perustuvaan koneeseen. Koska vanhaa järjestel-

mää ei ole suunniteltu tiedonkeruuta ajatellen, väylältä on saatavilla vain koneenohjauksen kannalta välttämättömät ohjausviestit ja anturien lukemat, joista ei voida päätellä kaikkea digitalisaation kannalta kiinnostavaa tietoa. Toisaalta koneen ohjaamiseen käytettävä CAN-väylä on huono tapa välittää ohjauksen kannalta epäoleellista tietoa. Väylän kuorma on pidettävä kohtuullisena, jottei ylikuormittuminen vaarantaisi koneenohjauksen toimintaa. Vanha järjestelmä ei tarjoa ohjausyksiköille muuta tapaa välittää tietoa toiminnastaan.

UKKO tarjoaa uutena ominaisuutena tuen erillisille turvaohjaimille (SPLC, Safety Programmable Logic Controller). SPLC valvoo ja ohjaa turvallisuuskriittisiä ominaisuuksia, kuten parkkijarrua ja hätäpysäytystä, ja varmistaa EU:n konedirektiivin ja teollisuusalakohtaisten turvallisuusstandardien vaatimusten toteutumista. SPLC:n ohjelmisto on toteutettu IEC 61508 -standardin [62] vaatimalla tavalla. UKKO:n turvaominaisuuksista on kerrottu lisää seuraavissa aliluvuissa.

UKKO myös yhtenäistää lastaus- ja kuljetuskoneiden ohjausjärjestelmän Sandvikin muiden ohjausjärjestelmien kanssa. Muun muassa Sandvikin porakoneet ovat jo pitkään perustuneet yhteiseen uudelleenkäytettävään ohjelmistoalustaan, jota tässä diplomityössä kutsutaan nimellä YKOA (Yleinen KoneenOhjausAlusta), ja johon UKKO perustuu myös. YKOA:sta on kerrottu lisää aliluvussa 5.4. YKOA mahdollistaa koneenohjausjärjestelmien osien uudelleenkäytön eri konetyyppien välillä, ja vähentää siten uusien, kaikkia konetyyppejä koskevien ominaisuuksien toteuttamiseen vaadittavaa työtä.

5.2 UKKO:n keskeiset ominaisuudet

Tunnelilouhinnassa kallioon räjäytetään ja kaivosporakoneella porataan mineraaliesiintymää seuraava tunneli. Kalliosta irronneet mineraalipitoiset kivenlohkareet kootaan puskukoneella varastokasoiksi, joista lastauskone käy myöhemmin hakemassa ne. Lastauskone kuljettaa kivenlohkareet suurella kauhallaan kymmenien tonnien kuormissa kuljetuskoneelle, joka kuljettaa ne edelleen jatkokäsiteltäväksi. Rakennettava prototyyppikone on tyypiltään lastauskone, mutta UKKO-koneenohjausjärjestelmä on tulevaisuudessa tarkoitus jatkokehittää tukemaan myös kuljetuskoneiden toteutusta.

UKKO tarjoaa kokoelman valmiiksi toteutettuja, konfiguraatiolla eri konemalleille sovitettavissa olevia ohjelmistokomponentteja koneenohjauksen, huoltotoimenpiteiden ja käyttöliittymän toteutukseen. UKKO määrittää yhdessä YKOA-ohjelmistoalustan (ks. aliluku 5.4) kanssa järjestelmän yleisen arkkitehtuurin ja suoritusvuon, jotta uusia konemalleja kehittävien asiakasprojektien ei tarvitse keksiä niitä uudelleen. UKKO tarjoaa siis lastauskoneille sovelluskehyksen, johon uusia konemalleja kehittävät asiakasprojektit voivat valita ja räätälöidä haluamansa ominaisuudet UKKO:n tarjoamasta valikoimasta, ja jonka ympärille asiakasprojektit voivat lisätä omia ohjelmistokomponenttejaan.

Koneenohjaukseen UKKO tarjoaa kaikille lastauskonemalleille yhteiset ominaisuudet, kuten kauhan, moottorin, vaihteiston, jarrujen ja valojen ohjauksen, suojaus-, kalibrointi- ja testaustoimintoja, radio- ja automaattiohjauksen sekä SPLC-integraation. Seuraavissa aliluvuissa on kuvattu tarkemmin näihin liittyviä vaatimuksia, ottamatta vielä kantaa siihen, miten ne kannattaa toteuttaa.

5.2.1 Parametrit, vaihdettavat osat ja optiot

Sandvikin lastauskoneet vaihtelevat keskenään koon ja käytettävissä olevien ominaisuuksien suhteen. Näistä syistä koneet voivat sisältää eri osia, kuten erikokoisen moottorin. Myös yhteiset osat, kuten kauha, voivat toimia eri konemalleissa eri tavalla. Edistyneemmät konemallit voivat sisältää lisäominaisuuksia, joita kutsutaan optioiksi. Samat optiot on mahdollista asentaa jälkikäteen myös perusmalleihin. Valinnaisten lisäominaisuuksien ja vaihdettavien koneen osien laajasta kirjosta huolimatta UKKO:n on sovelluttava kaikille konemalleille. Muuten koneenohjausjärjestelmästä olisi kehitettävä ja ylläpidettävä useaa eri versiota, mikä on työlästä, kallista ja on päinvastainen lopputulos kuin mihin UKKO:n kehittämisellä on pyritty.

UKKO:ssa ja muissa YKOA-pohjaisissa koneenohjausjärjestelmissä koneen ominaisuuksia voidaan muunnella niin kutsuttujen parametrien avulla, jotka ovat ohjausyksikön pysyvässä muistissa olevia muuttujia. Parametreja on mahdollista muokata YKOA:n valvojatasolla graafisen käyttöliittymän kautta. Osa parametreista säädetään valmiiksi tehtaalla, ja osa on koneen omistajan huoltohenkilöstön säädettävissä koneen elinkaaren aikana.

Esimerkki parametreilla säädettävästä ominaisuudesta: matkan ja nopeuden laskeminen. Prototyypissä koneen hetkellinen nopeus ja kulkema matka lasketaan voimansiirron vääntömuuntimen turbiiniakselin pyörimistaajuudesta. Anturin ilmoittama pyörimistaajuus ei suoraan kerro koneen matkanopeutta, sillä se riippuu myös koneen voimansiirtojärjestelmän ja asennettujen renkaiden ominaisuuksista. Vääntömuuntimen turbiiniakseli pyörittää vääntömuuntimen ulostuloakselia hammastussuhteen määräämällä nopeudella. Ulostuloakseli on kytketty kardaaniakseliin, joka pyörittää koneen pyöriä. Yhden pyörän kierroksen aikana kuljettu matka riippuu renkaan halkaisijasta. Matkan ja nopeuden laskeminen vaatii siten kaksi parametria: vääntömuuntimen hammastussuhteen ja renkaan halkaisijan. Hammastussuhde on asennetun voimansiirtojärjestelmän ominaisuus, eikä muutu koneen elinkaaren aikana, joten tämä parametri on asetettu jo tehtaalla. Koneeseen voi olla saatavilla useita erikokoisia renkaita, jotka on vaihdettava säännöllisesti koneen elinkaaren aikana, joten renkaan halkaisijaa kuvaavan parametrin on oltava koneen omistajan huoltohenkilöstön muutettavissa.

Esimerkki vaihdettavissa olevasta koneen osasta: moottori. Prototyyppi tuottaa tarvitsemansa voiman dieselmoottorilla, mutta UKKO:n suunnittelussa on varauduttava dieselmoottorien korvaamiseen tulevaisuudessa sähkömoottoreilla. UKKO:ssa moottorin on

oltava kohtuullisella vaivalla vaihdettavissa myös toisen malliseen dieselmoottoriin. Dieselmoottorit kehittyvät nopeasti, ja vaihdettavissa oleva moottori mahdollistaa moottorin päivittämisen uuteen, taloudellisempaan ja ympäristöystävällisempään malliin. Vaihdettavissa oleva moottori välttää myös niin kutsutun vendor lock -tilanteen, jossa työkoneen valmistaja on riippuvainen tietystä moottorivalmistajasta ja tietystä moottorimallista, altistaen itsensä hinnan noususta tai saatavuuden päättymisestä aiheutuville lisäkustannuksille ja tuotannon pysähdyksille. Kullakin moottorimallilla ECU:n kommunikaatiorajapinta voi olla yleisestä J1939-standardista huolimatta hieman erilainen, joten uuden moottorimallin tukeminen vaatii muutoksia UKKO:n toteutukseen. Muutosten suuruus on kuitenkin minimoitu abstraktioiden, avoin/suljettu -periaatteen ja yhden vastuualueen periaatteen avulla.

Esimerkki optiosta: automaattinen kauhan ravistus. Kun lastauskone pudottaa lastinsa kuljetuskoneen lavalle, kauhaan saattaa jäädä jumiin maa-ainesta. Jotta loputkin kuormasta saadaan pois kauhasta, kauhaa on ravistettava voimakkaasti. Lastauskoneen perusmalleissa operaattorin on tehtävä tämä itse heiluttamalla kauhan ohjaussauvaa terävästi edestakaisin. Kaikkiin lastauskoneisiin on kuitenkin saatavilla lisäominaisuutena automaattinen kauhan ravistus, joka tekee ravistusliikkeen automaattisesti yhdellä napin painalluksella parantaen koneen käyttömukavuutta. Vaikka koneessa ei olisi automaattista kauhan ravistusta käytössä, optio on olemassa koneenohjausjärjestelmän ohjelmistossa, joten ominaisuuden käyttöönotto ei vaadi ohjelmiston päivittämistä. Sandvikin valtuuttama huoltohenkilö aktivoi optiot valmiiksi tehtaalla tai vieraillessaan asiakkaan luona, jos ominaisuus otetaan käyttöön jälkikäteen. Jotkin optiot vaativat ylimääräistä laitteistoa, joka sekin voidaan asentaa valmiiksi tehtaalla tai jälkikäteen.

5.2.2 Vaihtoehtoiset ohjausmoodit

Työkoneissa on usein vaihtoehtoisia ohjausasemia tai ohjausmoodeja, jotka mahdollistavat koneen ohjaamisen fyysisesti eri paikoista [7, s. 330-335]. Sandvikin lastauskoneissa on kolme vaihtoehtoista ohjausmoodia: paikallinen ohjausmoodi, radio-ohjausmoodi ja automaatio-ohjausmoodi. EU:n konedirektiivi vaatii, että vain yksi ohjausmoodi saa olla kerrallaan käytössä [10]. Prototyypissä direktiivin vaatimuksen toteutuminen on varmistettu toteuttamalla ohjausmoodin valinta fyysisellä vipukytkimellä, ja validoimalla valittu ohjausmoodi SPLC:n ohjelmistossa.

Kun paikallinen ohjausmoodi on valittuna, operaattori istuu koneen hytissä ja ohjaa konetta hytissä olevilla ohjaussauvoilla, napeilla ja polkimilla. Paikallinen ohjausmoodi on saatavilla kaikille konemalleille, ja on koneen oletusarvoinen ohjausmoodi.

Toisinaan lastauskoneen on mentävä kaivoksessa paikkoihin, jonne ihmiset eivät voi sortumisvaaran vuoksi mennä. Tällöin kone voidaan asettaa radio-ohjausmoodiin, jolloin konetta voidaan ohjata radio-ohjaimella turvalliselta alueelta käsin. Radio-ohjaus on optio, joka aktivoinnin lisäksi vaatii perusmallista puuttuvan radiovastaanottimen (RRC,

katso kuva 17) asentamisen koneeseen. Optioparametrin asetus ja RRC:n asennus voidaan tehdä valmiiksi tehtaalla tai jälkikäteen Sandvikin valtuuttaman asentajan toimesta.

Automaatio-ohjausmoodissa lastauskonetta ohjataan maanpinnalla sijaitsevasta valvomosta käsin. Valvomossa voi olla ihminen antamasta komentoja koneelle, tai komennot voivat tulla automaatiojärjestelmältä, mistä ohjausmoodin nimi tulee. Komennot välitetään kaivokseen asennettujen tukiasemien kautta koneessa olevalle automaatiotietokoneelle (katso kuva 17), joka välittää komennot eteenpäin varsinaiselle koneenohjausjärjestelmälle. Kuten radio-ohjausmoodi, automaatio-ohjausmoodi on optio, jonka käyttöönotto vaatii Sandvikin valtuuttaman asentajan aktivoimaan option ja asentamaan tarvittavan lisälaitteiston.

Riippumatta valitusta ohjausmoodista koneenohjausjärjestelmä reagoi annettuihin komentoihin aina samalla tavalla. Yhtenäisen, ohjausmoodista riippumattoman ohjauksen toteutusta on käsitelty aliluvussa 5.5.6.

5.2.3 Turvaominaisuudet

Kuten luvussa 2.4 kerrottiin, laki ja sovellusalakohtaiset vapaaehtoiset turvallisuusstandardin asettavat erityisiä vaatimuksia kaivoslastauskoneiden turvallisuudelle. Prototyypissä on kaksi SPLC:tä, jotka mahdollistavat ja varmistavat joidenkin turvallisuusvaatimusten toteutumisen. Loput vaatimukset on toteutettu ohjelmiston standardipuolella tai laitteisto- ja mekaniikkasuunnittelulla. Ohjelmiston standardipuoleksi kutsutaan sitä osaa ohjelmistosta, jota ei ole toteutettu, varmennettu ja dokumentoitu turvallisuuskriittiseltä ohjelmistolta standardissa IEC 61508 vaaditulla tavalla. SPLC valvoo kaikkia hätäpysäytyskomentoja, parkkijarrun toimintaa, hytin oven kiinni pysymistä, ohjausmoodin valintaa ja kauko-ohjauksen langattoman yhteyden kuntoa.

EU:n konedirektiivi vaatii, että koneen jokaisella ohjausasemalla on vähintään yksi selvästi merkitty hätäpysäytyspainike. Sandvikin lastauskoneissa ohjausasemia on kolme (hytti, radio-ohjain ja automaatio-ohjaus), ja lisäksi koneessa on useita ylimääräisiä hätäpysäytyspainikkeita vaarallisiksi katsotuissa paikoissa. Kaikki koneessa olevat hätäpysäytyspainikkeet, ovat kytketty suoraan SPLC:hen. Radio- ja automaatiovastaanottimet sisältävät hätäpysäytysulostulot, jotka on yhdistetty suoraan SPLC:n vastaaviin sisäänmenoihin.

Hätäpysäytyskäskyn saadessaan SPLC katkaisee sähköt koneen liikkuvista osista, sammuttaa moottorin ja kytkee parkkijarrun päälle. SPLC ilmoittaa hätäpysäytyksestä CAN-väylällä myös standardipuolen ohjausyksikölle, mutta tämän tarkoitus on vain estää näennäisistä sähkövioista ja puuttuvista moduuleista aiheutuva hämmennys ja tarpeettomat hälytykset, ilmoittaa hätäpysäytystilasta operaattorille ja kirjata tapahtuma lokiin. Stan-

dardipuolen ohjelmisto ei pysty hätäpysäytyksen aikana ohjaamaan yhtäkään koneen liikkuvista osista, eikä se voi peruuttaa tai kuitata hätäpysäytystilaa. Hätäpysäytystilan kuittaaminen on myös SPLC:n valvonnassa.

SPLC ohjaa parkkijarrua, ja valvoo sen tilaa. Parkkijarrun ollessa päällä, SPLC kytkee sähköt pois niiltä toimilaitteilta, joiden käyttö ei ole sallittua parkkijarrun ollessa päällä. Tällaisia toimilaitteita ovat käytännössä kaikki koneen liikkuvat osat, pois lukien moottorin käyminen. Näin standardipuolen ohjelmistoa estetään vahingossakaan aiheuttamasta vaaraa ohjaamalla kiellettyjä toimilaitteita. Prototyyppikoneen parkkijarrun hytissä oleva painike on kytketty suoraan SPLC:n sisäänmenoon. Standardipuolen ohjausyksikkö voi lisäksi pyytää parkkijarrun vapauttamista tai päälle kytkemistä CAN-väylän kautta, ja etäohjauksen parkkijarrun ohjauspyynnöt välitetään SPLC:lle tätä kautta. Standardipuolen ohjelmisto ei voi suoraan ohjata parkkijarrua, eikä SPLC:n ole pakko suostua parkkijarrun vapautukseen, mikäli se ei jostakin syystä ole turvallista.

Operaattorin suojelemiseksi koneen hytin oven on oltava suljettuna aina, kun kone voi liikkua. Hytin ovessa on oven tilan tunnistava kytkin, joka on yhdistetty SPLC:n sisäänmenoon. SPLC pitää parkkijarrun päällä niin kauan, kuin hytin ovi on auki. Jos hytin ovi aukeaa kesken ajon, SPLC hidastaa ja pysäyttää koneen hallitusti, ja kytkee parkkijarrun päälle.

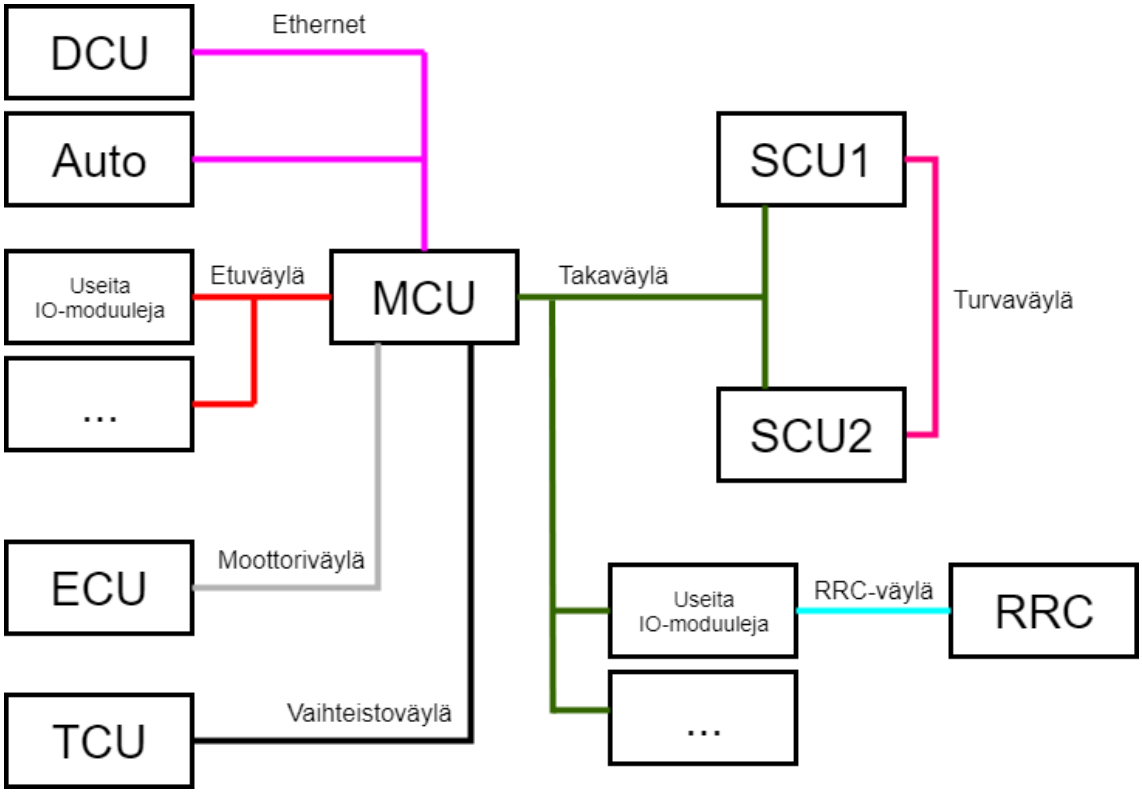
Prototyyppikoneen ohjausmoodin valitseva vipukytkin on kytketty SPLC:n sisäänmenoihin. SPLC valvoo, että vipukytkimessä on vain yksi tila valittuna. Standardipuolen ohjelmisto saa tiedon valitusta moodista SPLC:ltä CAN-väylän kautta. SPLC kytkee radiovastaanottimesta ja automaatiotietokoneesta sähköt pois, kun niitä käyttävä ohjausmoodi ei ole valittuna, estäen niiden toiminnan väärässä ohjausmoodissa.

Koneen etäohjaus edellyttää langatonta tiedonsiirtoa, jota pidetään epäluotettavana tiedonsiirtotapana. Etäohjaimen yhteyttä koneeseen on valvottava jatkuvasti, ja kone on pysäytettävä, jos yhteys katkeaa. Etäohjaimet lähettävät säännöllisesti niin kutsuttua sykeviestiä (heartbeat), mistä vastaanotin tietää lähettimen olevan päällä, ja yhteyden toimivan. Prototyyppikoneen radiovastaanottimessa ja automaatiotietokoneessa on normaalista sykkeestä ilmoittava ulostulo, joka on yhdistetty SPLC:n sisäänmenoihin. Jos langaton yhteys etäohjaimeen katkeaa, SPLC pysäyttää koneen ja kytkee parkkijarrun päälle. Kone voi jatkaa toimintaansa vasta kun yhteys on palautunut ja etäohjain on erikseen pyytänyt parkkijarrun vapauttamista.

5.3 Järjestelmäarkkitehtuuri

Prototyyppi koostuu useista keskenään verkotetusta ohjausyksiköstä ja IO-moduulista. Moduulit kommunikoivat keskenään käyttäen useaa eri kommunikaatioväylää, jotka ovat keskenään eri tyyppisiä ja käyttävät eri protokollia. Prototyyppikoneen laitteistomoduulit ja niitä yhdistävät väylät on esitetty yksinkertaistetussa kuvassa 17. Moduulien lyhenteet

on listattu taulukossa 9. Kommunikaatioväylät ja niiden käyttämät protokollat on listattu taulukossa 10.



Kuva 17. *Prototyyppikoneen yksinkertaistettu rakenne.*

Taulukko 9. *Prototyyppikoneen moduulien merkintöjen selitykset.*

Merkintä	**Selitys**
ECU	Engine Controller Unit, moottorinohjausyksikkö
TCU	Transmission Controller Unit, vaihteistonohjausyksikkö
MCU	Master Controller Unit, pääohjausyksikkö
DCU	Display Controller Unit, kosketusnäyttötietokone
RRC	Radio Remote Controller, kolmannen osapuolen radiovastaanotin
SCU1	Safety Controller Unit 1, turva-PLC
SCU2	Safety Controller Unit 2, turva-PLC
Auto	Automaatiotietokone

Taulukko 10. *Prototyyppikoneen kommunikaatioväylät.*

Nimi (ja väri) kuvassa 17	**Väylän tyyppi**	**Protokolla**	**Nopeus**
Ethernet (lila)	Ethernet	TCP/IP	100 Mbps
Etuväylä (punainen)	CAN	CANopen	500 kbps
Takaväylä (vihreä)	CAN	CANopen	500 kbps
Vaihteistoväylä (musta)	CAN	SAE J1939	250 kbps
Moottoriväylä (harmaa)	CAN	SAE J1939	250 kbps
RRC-väylä (turkoosi)	CAN	CANopen	250 kbps
Turvaväylä (vaaleanpunainen)	CAN	CANopenSafety	500 kbps

Vaikka järjestelmää voidaan usean verkotetun ohjausyksikkönsä vuoksi kutsua hajautetuksi koneenohjausjärjestelmäksi, kaikki sovelluskohtainen standardipuolen koneenohjauslogiikka on keskitetty MCU-moduulille. Tällä perusteella järjestelmää voi kutsua myös keskitetyksi koneenohjausjärjestelmäksi. MCU:n lisäksi vain DCU, SCU1 ja SCU2 sisältävät sovelluskohtaista ohjelmistoa. DCU:n ohjelmisto toteuttaa YKOA:n valvojatason, joka ei ole välttämätön koneenohjauksen kannalta, ja sitä on siksi käsitelty vain lyhyesti tässä diplomityössä. SCU1 ja SCU2 ovat SPLC-yksiköitä. Niiden ohjelmisto on toteutettu erillisenä projektina, minkä vuoksi niiden suunnittelua tai toteutusta ei käsitellä tässä diplomityössä. SPLC-yksiköiden merkitys järjestelmässä on selitetty aliluvussa 5.2.3. Seuraavat aliluvut keskittyvät MCU:n ohjelmiston suunnitteluun ja toteutukseen.

5.4 YKOA-ohjelmistoalusta

YKOA on Sandvikin kehittämä ohjelmistoalusta omille koneenohjausjärjestelmilleen. YKOA on ollut käytössä muun muassa maanpäällisten - ja maanalaisten kaivosporakoneiden koneenohjausjärjestelmien toteutuksessa, ja UKKO:n myötä sitä on mahdollista käyttää myös uusien lastaus- ja kuljetuskoneiden toteutuksissa. YKOA:n tarkoituksena on tarjota valmis ohjelmistokehys, jonka ympärille konetyyppikohtaiset koneenohjausjärjestelmät voidaan toteuttaa.

YKOA on jakautunut kahteen osaan: koneenohjaukseen (MC, Machine Control) ja valvojatasoon. Kahtiajako noudattaa koneenohjausjärjestelmissä usein käytettyä suunnittelumallia: erillistä reaaliaikaa (separate real-time) [7, s. 237-243]. Suunnittelumallissa kovia reaaliaikavaatimuksia sisältävät toiminnot on toteutettu eri laitteistolla kuin vähemmän ajoituskriittiset toiminnot. Tällä tavalla kaikki toiminnot saavat suorittimelta riittävästi suoritusaikaa häiritsemättä toisiaan. Samalla ohjausyksiköllä toteutettuna reaaliaikaiset toiminnot olisivat etusijalla, ja muut toiminnot, kuten graafinen käyttöliittymä, saisivat liian vähän suoritinaikaa toimiakseen sulavasti.

Valvojataso toteuttaa järjestelmän vähemmän kriittiset toiminnot, joilla ei ole kovia reaaliaikavaatimuksia. Valvojataso helpottaa koneen huoltoa ja diagnosointia sekä parantaa käyttäjäkokemusta, mutta ei ole välttämätön koneenohjauksen toiminnalle. Se tarjoaa muun muassa graafisen käyttöliittymän anturilukemien, diagnostiikkanäkymien ja tapahtumalokien tarkasteluun, ohjattuja kalibrointi- ja testaustoimintoja, järjestelmän ohjelmiston päivittämisen sekä koneen ominaisuuksia säätelevien parametrien säätämisen ja varmuuskopioinnin. Prototyyppikoneessa valvojatason laitteistona toimii 12-tuumainen kosketusnäyttötietokone, mutta YKOA:n valvojataso on alustariippumaton, ja asennettavissa myös eri kokoisille ja usean eri valmistajan näytölle. Tässä diplomityössä keskitytään koneenohjauspuolen toteutukseen ja suunnitteluun, eikä valvojatasoa käsitellä tämän perusteellisemmin.

YKOA:n MC-taso sisältää järjestelmän reaaliaikaiset toiminnot, eli kaikkien koneen liikkuvien osien ohjaamisen ja koneen ohjainten tarkkailun. YKOA:n MC-taso on täysin itsenäinen, eikä vaadi valvojatasoa toimiakseen. Se toimii alustana koneenohjausohjelmistolle, joka on toteutettu koneen kehitystiimin määrittelemien niin kutsuttujen applikaatioiden avulla. Selvyyden vuoksi YKOA:n tarjoamaa valmista koneenohjausalustaa kutsutaan jatkossa MC-alustaksi, ja MC-alustan ja applikaatioiden muodostamaa koko koneenohjausjärjestelmää MC-tasoksi.

5.4.1 YKOA:n MC-alusta

YKOA:n MC-alusta on ohjelmistoalusta yksittäisen ohjausyksikön toimintalogiikan toteuttamiselle. Järjestelmässä voi olla useita MC-ohjausyksiköitä, jotka viestivät keskenään kommunikaatioväylien kautta. Prototyyppikoneessa MC-ohjausyksiköitä on vain yksi (MCU, katso kuva 17). MC-alusta toteuttaa sovelluksesta riippumattomat yleiset toiminnot. MC-alusta lähettää, vastaanottaa ja tulkitsee kommunikaatioväylien viestejä, jakaa suoritusaikaa applikaatioille ja valvoo järjestelmän reaaliaikavaatimusten toteutumista. Se valvoo myös MC-ohjausyksikön hallinnoimien väylien muita solmuja, ja diagnosoi kommunikaatiovirheitä niiden välillä.

MC-alusta toteuttaa luvussa 4.4 kuvatun IO-abstraktion ohjausyksikön väylille ja fyysisille IO-liitännöille yksikkömuunnoksia lukuun ottamatta. MC-alusta tukee CAN- ja Ethernet-väyliä, ja korkean tason CAN-protokollista CANopen- ja SAE J1939-protokollia. MC-ohjausyksikkö on yleensä Ethernetin kautta yhteydessä valvojatasoon, ja CAN-väylien kautta IO-moduuleihin ja muihin ohjausyksiköihin.

MC-alustan IO-abstraktion viestialkiota kutsutaan signaaliksi. Signaali kuvaa yksittäistä väylän tai fyysisen IO:n tietoalkiota, kuten CANopen PDO-viestissä vastaanotettua ohjaussauvan poikkeutusta, valvojatasolle Ethernetin kautta lähetettävää diagnostiikkatietoa tai ohjausyksikön ulostulon ohjausvirtaa. Kullakin signaalilla on luvun 4.4 mukaisesti yksilöllinen tunniste, arvo ja validiteetti.

Koneenohjauksen konemallikohtainen logiikka toteutetaan koneen kehitysprojektin itse määrittelemillä ohjelmilla, joita kutsutaan applikaatioiksi. Applikaatio noudattaa Super Loop -rakennetta [7, s. 12-31], ja MC-alusta suorittaa kutakin applikaatiota yhden kierroksen jokaisella omalla suorituskierroksellaan. Yhdellä MC-ohjausyksiköllä voi periaatteessa olla rajoittamaton määrä applikaatioita. Kullakin MC-ohjausyksiköllä voi kuitenkin olla vain yksi erityinen Input-applikaatio, joka suoritetaan applikaatioista ensimmäisenä, ja vain yksi erityinen Output-applikaatio, joka suoritetaan applikaatioista viimeisenä. Applikaatioiden toteutus ja vuorovaikutus keskenään on selitetty tarkemmin seuraavassa aliluvussa.

MC-alustan suoritus on syklinen: MC-alusta suorittaa kaikki toimintonsa konfiguroitavan kiertoajan intervalleissa. Prototyyppikoneessa tämä kiertoaika on 10 ms. MC-alustan joka

kierroksella toistamat toiminnot ovat vastaanotettujen väyläviestien käsittely, kaikkien applikaatioiden suorittaminen yhden kierroksen verran ja lähetettävien väyläviestien käsittely. Kiertoaika on ehdoton, eikä toimintojen suorittaminen saa kestää sitä kauempaa, koska se vaarantaisi kovien reaaliaikavaatimusten toteutumisen ja koneen turvallisuuden. MC-alusta tarkkailee jatkuvasti omaa todellista kiertoaikaansa, ja pysäyttää järjestelmän, mikäli kiertoaika ylittyy. Ominaisuus on koneenohjausjärjestelmissä yleinen, ja sitä kutsutaan vahtikoiraksi (watchdog) [7, s. 101-106].

Taulukossa 11 on esitetty yksi MC-alustan suorituskierros. Toiminnot suoritetaan samassa järjestyksessä kuin ne ovat taulukossa. Vaihtelevan väyläkuorman vuoksi MC-alustan kierrosaikaan on varattava hieman ylimääräistä joutoaikaa, jottei hetkellinen piikki saapuvien viestien määrässä aiheuttaisi kiertoajan ylitystä. Mikäli joutoaikaa jää jäljelle pakollisten toimintojen suorittamisen jälkeen, MC-alusta käyttää tämän ajan pieniprioriteettisten taustaprosessien suorittamiseen, kuten lokiviestien tallentamiseen.

Taulukko 11. *YKOA:n MC-alustan suorituskierroksen toiminnot suoritusjärjestyksessä.*

Saapuneiden viestien käsittely	Input-app.	App1, App2, ...	Output-app.	Lähetettävien viestien käsittely	joutoaika
Kiertoaika (esim. 10 ms)					

5.4.2 Applikaatiot

Applikaatiot sisältävät kaiken varsinaisen sovelluskohtaisen koneenohjauslogiikan. Kuten edellä todettiin, MC-alusta suorittaa kutakin applikaatiota kerran jokaisen kiertoajanjakson aikana. Kullakin MC-ohjausyksiköllä voi olla yksi input-applikaatio, yksi output-applikaatio, ja periaatteessa rajoittamaton määrä muita applikaatioita. Rajallinen kiertoaika rajoittaa applikaatioiden lukumäärää käytännössä. Itse MC-alusta on kirjoitettu C-kielellä, mutta se tukee myös C++- IEC 61131-3- ja Simulink -kielillä toteutettuja applikaatioita.

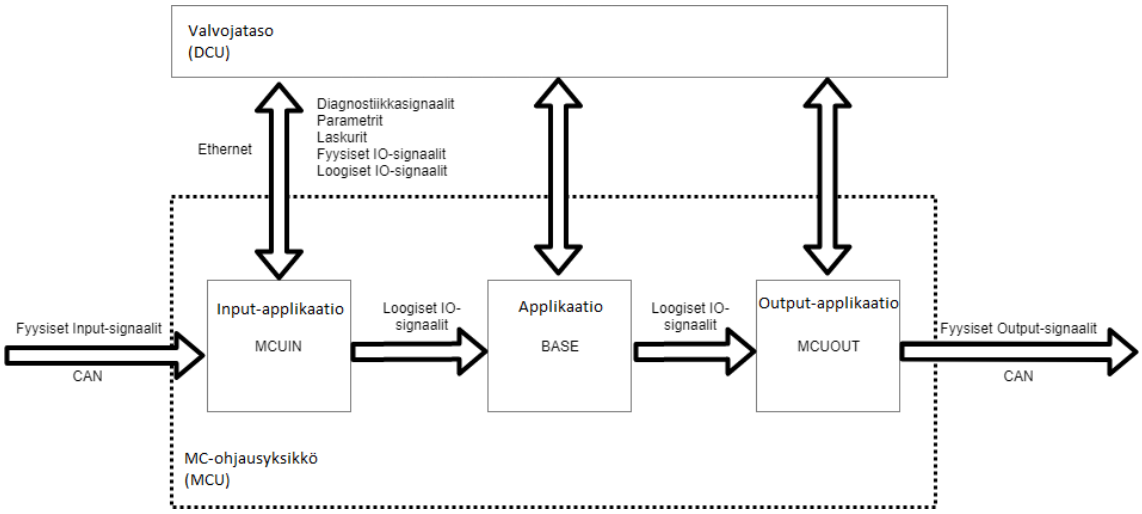
Kaikilla applikaatioilla on niiden tyypistä riippumatta yhdenmuotoinen rajapinta MC-alustan kanssa, jota kutsutaan prosessikuvaksi (MCPI, Machine Control Process Image). Prosessikuva sisältää seuraavat määrittelyt:

1. Sisäänmenosignaalit, joiden kautta applikaatio vastaanottaa lähtötietonsa antureilta, muilta ohjausyksiköiltä tai applikaatioilta. MC-alustan konfiguraatiossa on määritelty kunkin applikaation sisäänmenosignaalit ja niiden lähteet. Sisäänmenosignaalin lähteenä voi olla ohjausyksikön fyysinen sisäänmeno, kommunikaatioväylältä vastaanotettu signaali tai jonkin toisen applikaation ulostulosignaali. Sisäänmenosignaalilla voi olla vain yksi lähde, eikä se voi muuttua kesken suorituksen.

2. Ulostulosignaalit, joiden kautta applikaatio välittää laskentansa tulokset toimilaitteille, muille ohjausyksiköille tai applikaatioille. MC-alustan konfiguraatiossa on määritelty kunkin applikaation ulostulosignaalit, ja niiden kohteet. Ulostulosignaalilla voi olla useita kohteita, ja ne voivat olla ohjausyksikön fyysisiä ulostuloja, kommunikaatioväylälle lähetettäviä signaaleja tai muiden applikaatioiden sisäänmenosignaaleja.
3. Parametrit, jotka ovat ohjausyksikön haihtumattomaan muistiin tallennettuja muuttujia, ja jotka säätelevät applikaation ominaisuuksia. Parametrien avulla koneenohjausjärjestelmä saadaan mukautettua eri konemalleille sopivaksi, kuten edellisissä aliluvuissa on esimerkein selitetty. Parametreille on MC-alustan konfiguraatiossa määritelty arvoalue ja oletusarvo, mutta valvojataso voi muuttaa parametrin arvoa myöhemmin. Uusi arvo tallentuu välittömästi ohjausyksikön haihtumattomaan muistiin, ja säilyttää uuden arvonsa järjestelmän uudelleenkäynnistyksen jälkeenkin.
4. Diagnostiikkasignaalit, joiden kautta applikaatio voi välittää valvojatasolle hyödyllistä diagnostiikkatietoa, ilmoittaa lukitustilanteista tai poikkeustapauksissa vastaanottaa komentoja valvojatasolta. Diagnostiikkasignaalit eivät tallennu ohjausyksikön pysyvään muistiin, vaan ne nollautuvat aina ohjausyksikön sammuessa.
5. Laskurit, joiden avulla applikaatio voi tallettaa usein päivittyviä historiatietoja, kuten koneen trippimittarin lukeman, ohjausyksikön haihtumattomaan muistiin. Valvojataso voi lukea ja nollata laskurit. Toisin kuin parametrit, laskurien arvo ei päivity välittömästi ohjausyksikön haihtumattomaan muistiin, vaan MC-alustan konfiguraatiossa määritellyn intervallin välein, mikäli laskurin arvo on tänä aikana muuttunut. Tällä pidennetään fyysisten muistipaikkojen käyttöikää, jota jokainen kirjoitusoperaatio lyhentää.
6. Hälytykset, joilla applikaatio voi ilmoittaa operaattorin välitöntä huomiota vaativista virhetilanteista tai tapahtumista valvojatasolle. Tällaisia virhetilanteita ovat muun muassa epänormaalit anturilukemat, kommunikaatiovirheet väylillä, sähköiset viat, virheet ohjauksessa ja vaaralliset operaattorin virheet. MC-alusta aktivoi itse hälytykset signaalien normaalien raja-arvojen ylityksistä, sähkövioista ja kommunikaatiovirheistä. Applikaatioiden vastuulle jää ilmoittaa ohjauksen tai operaattorin virheistä, esimerkiksi koneen ajamisesta jarrut laahaten.

Kuva 18 havainnollistaa, miten applikaatiot vuorovaikuttavat ympäristönsä kanssa prosessikuvansa avulla. Prototyyppikoneessa on vain yksi MC-ohjausyksikkö, MCU. MCU:lla on kolme applikaatiota: input-applikaatio (MCUIN), output-applikaatio (MCUOUT) ja varsinaisesta koneenohjauslogiikasta vastaava applikaatio (BASE). MC-alusta suorittaa applikaatiot järjestyksessä: MCUIN, BASE, MCUOUT. Kukin applikaatio päivittää ulostulosignaalinsa joka suorituskierroksella, ja uusi arvo on jo samalla suorituskierroksella myöhemmin suoritettavien applikaatioiden käytössä. MCU:n sisäänmenoonsa saama heräte aiheuttaa siten vasteen MCU:n ulostuloissa heti seuraavan MC-alustan suorituskierroksen jälkeen. Pahimmillaan viive herätteestä vasteeseen on siten kaksinkertainen kiertoaikaan nähden, eli prototyypin tapauksessa 20 ms. Kuvasta ilmenee,

miten applikaatiot vaikuttavat toisiinsa loogisten IO-signaalien kautta, mutta eivät näe muita osia toistensa prosessikuvasta. YKOA:n valvojataso sen sijaan näkee kaikkien applikaatioiden koko prosessikuvan.



Kuva 18. *MCU:n applikaatiot ja niiden vuorovaikutukset.*

MCUIN- ja MCUOUT-applikaatiot muodostavat luvussa 4.3 kuvatun koko järjestelmän HAL:n muille applikaatioille. MCUIN-applikaation pääasiallinen tehtävä on siis muuntaa kommunikaatioväyliltä tai fyysisistä sisäänmenoista saadut raaka-arvot BASE-applikaatiolle mielekkääseen muotoon (SI-yksiköiksi). MCUOUT-applikaation pääasiallinen tehtävä vastaavasti on muuntaa BASE-applikaation tuottamat ohjaussignaalit toimilaitteiden vaatimiksi raaka-arvoiksi. Suurin osa MCUIN- ja MCUOUT-applikaatioiden koodista on sähkösuunnittelijan tekemän IO-listan perusteella automaattisesti generoitua C++-koodia. IO-listasta käy ilmi koko järjestelmän kaikki fyysiset sisäänmenot ja ulostulot, sekä muunnoskaavat raaka-arvoista SI-yksiköiksi. MCUIN- ja MCUOUT-applikaatioiden koodi generoidaan uudelleen jokaiselle eri konemallille. Seuraavissa aliluvuissa syvennytään siksi vain BASE-applikaation, eli varsinaisen koneenohjauslogiikan, uudelleenkäytön mahdollistavaan suunnitteluun ja toteutukseen.

5.5 UKKO:n uudelleenkäytettävä koneenohjauslogiikka

Tässä aliluvussa käsitellään UKKO:n varsinaisen koneenohjauslogiikan suunnittelua ja toteutusta. Aliluvussa käydään läpi ohjauslogiikan toteuttavan applikaation toteutusteknologioiden ja ohjelmointiparadigmojen valintaperusteet, applikaation rakenne ja keskeiset uudelleenkäyttöä tukevat suunnitteluperiaatteet.

Tässä diplomityössä ei ole tarkoitus käydä läpi applikaation koko arkkitehtuuria, vaan esille on nostettu joitakin uudelleenkäytön kannalta tärkeimpiä ratkaisuja. Suunnitteluratkaisuja on havainnollistettu esimerkein. Esimerkeissä moduulien ja rajapintojen nimet eivät vastaa todellisuutta, vaan todelliset nimet on selkeyden vuoksi korvattu esimerkin kannalta paremmin kuvaavilla, suomenkielisillä nimillä. Esimerkit ovat pelkistettyjä, ja

niistä on jätetty pois epäoleellisia yksityiskohtia. Esimerkit vastaavat kuitenkin periaatteen tasolla oikeaa järjestelmää.

5.5.1 Toteutustekniikoiden ja ohjelmointiparadigmojen valinta

Koneenohjauslogiikasta vastaava BASE-applikaatio on toteutettu IEC-61131-3 ST- ja FBD-kielillä CODESYS 3.5 -ohjelmointiympäristössä. Valinta tehtiin, koska CODESYS-sovellusten kehittäminen ja testaaminen on helpompaa kuin vastaavien C- tai C++ -sovellusten kehittäminen. CODESYS tarjoaa helpon tavan päivittää ohjelmisto kohdelaitteelle jopa kesken ohjelman suorituksen, ja se tarjoaa selkeän käyttöliittymän ohjelman debuggaamiseen. Toistaiseksi vastaavia työkaluja ei ole olemassa helpottamaan MC-alustan C- ja C++-applikaatioiden kehitystyötä. Kaupallisena tuotteena CODESYS:iin liittyviin ongelmiin voi saada sen kehittäjältä varmemmin tukea kuin C- ja C++-foorumeilta.

CODESYS 3.5-versio mahdollistaa olio-ohjelmoinnin IEC 61131-3 -ohjelmointikielillä, ja se on siksi valittu vanhemman ja yleisemmin käytetyn CODESYS 2.3-version sijaan. Olio-ohjelmointi on valittu BASE-applikaation pääasialliseksi ohjelmointiparadigmaksi, koska se mahdollistaa abstraktien rajapintojen käytön. Abstraktien rajapintojen ansiosta ohjelmistossa on voitu välttää tiukkoja riippuvuussuhteita ohjelmistokomponenttien välillä, mikä on kriittistä ohjelmiston uudelleenkäytön kannalta. Mikä tahansa komponentti voidaan korvata toisella ilman, että siitä riippuvien muiden komponenttien toteutusta tarvitsee muuttaa. Yksittäisten ohjelmistokomponenttien sisäisessä toteutuksessa on mahdollisuuksien mukaan sovellettu myös funktionaalista ohjelmointia, vaikka IEC-61131-3 -ohjelmointikielet eivät tue puhdasta funktionaalista paradigmaa. Funktionaalisen paradigman soveltamisella on pyritty koodin selkeyteen ja helppoon debuggaukseen.

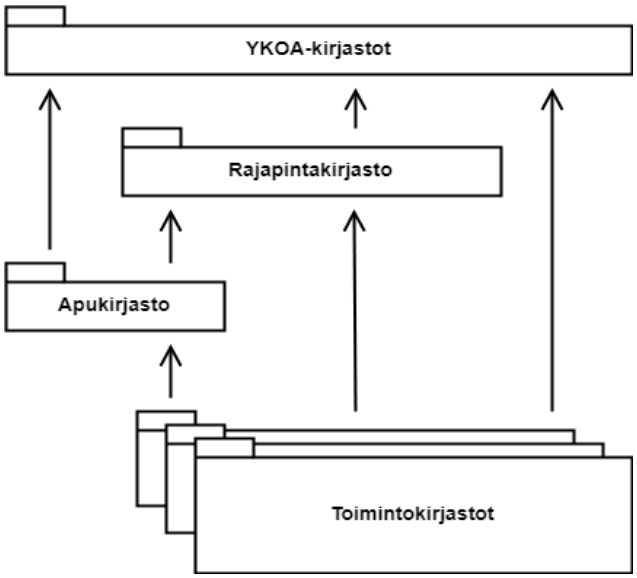
IEC 61131-3 ST- ja FBD-kielet soveltuvat eri tarkoituksiin applikaation toteutuksessa. FBD sopii graafisen esitysmuotonsa vuoksi suurten kokonaisuuksien ja niiden välisten riippuvuuksien esittämiseen. FBD on erityisen mukava ohjelmointikieli debuggauksen kannalta. FBD-diagrammista näkee nopealla silmäyksellä funktiolohkojen vaikutukset toisiinsa, ja niiden hetkelliset sisäänmeno- ja ulostuloparametrien arvot. Tekstimuotoinen ST-kieli sen sijaan sopii paremmin runsaasti ehtolauseita sisältävän matalan tason logiikan toteuttamiseen, koska haarautuvan logiikan esittämien FBD-kielellä vaatii luettavuutta haittaavien hyppykäskyjen käyttämistä. Muita IEC 61131-3:n ohjelmointikieliä ei ole käytetty koodin yhdenmukaisena pitämiseksi, ja koska IL- LD- ja SFC-kielet soveltuvat huonosti olio-ohjelmointiin.

5.5.2 BASE-applikaation rakenne

BASE-applikaatio on CODESYS-sovellus, joka koostuu varsinaisesta CODESYS-projektista ja siihen liitetyistä kirjastoista. UKKO ja YKOA tarjoavat suurimman osan kirjastoista, ja projekti on UKKO:oon perustuvien koneiden kehittäjien vapaasti muokattavissa. Koneiden kehittäjät voivat myös vapaasti toteuttaa omia kirjastojaan.

UKKO:n kirjastojako muistuttaa koneen fyysistä rakennetta. Kutakin koneen ohjattavaa osaa, esimerkiksi puomia, moottoria ja jarruja, vastaa yksi kirjasto. Koneen toimintoja toteuttavia kirjastoja kutsutaan jatkossa toimintokirjastoiksi. UKKO tarjoaa koneen toimintojen toteutuksessa auttavan, yleiskäyttöisiä funktioita ja funktiolohkoja sisältävän ja UKKO:n sisäistä koodin uudelleenkäyttöä tukevan apukirjaston. UKKO tarjoaa myös rajapintakirjaston, joka määrittelee toimintokirjastojen komponenttien abstraktit rajapinnat, joiden välityksellä komponentit liittyvät toisiinsa. Kaikki UKKO:n kirjastot voivat käyttää toteutuksessaan myös YKOA-kirjastoja, jotka tarjoavat rajapinnan YKOA:n MC-alustan palveluihin ja siten fyysisen koneen ohjaamiseen.

Kirjastoilla on tiukka keskinäinen hierarkia: Toimintokirjastot voivat olla riippuvaisia apukirjastosta ja rajapintakirjastosta, apukirjasto on riippuvainen rajapintakirjastosta, ja rajapintakirjasto on riippuvainen vain YKOA-kirjastoista. Hierarkia on yksisuuntainen. Toimintokirjastot eivät ole koskaan suoraan riippuvaisia toisistaan, vaan riippuvuuksien kääntämisen periaatteen mukaisesti ainoastaan rajapintakirjaston rajapintojen kautta. Tiukalla kirjastohierarkialla vältetään toimintokirjastojen tiukat riippuvuussuhteet, jotka estäisivät yksittäisten toimintokirjastojen uudelleenkäytön tai korvaamisen toisella. Kirjastojen hierarkia on esitetty kuvassa 19.

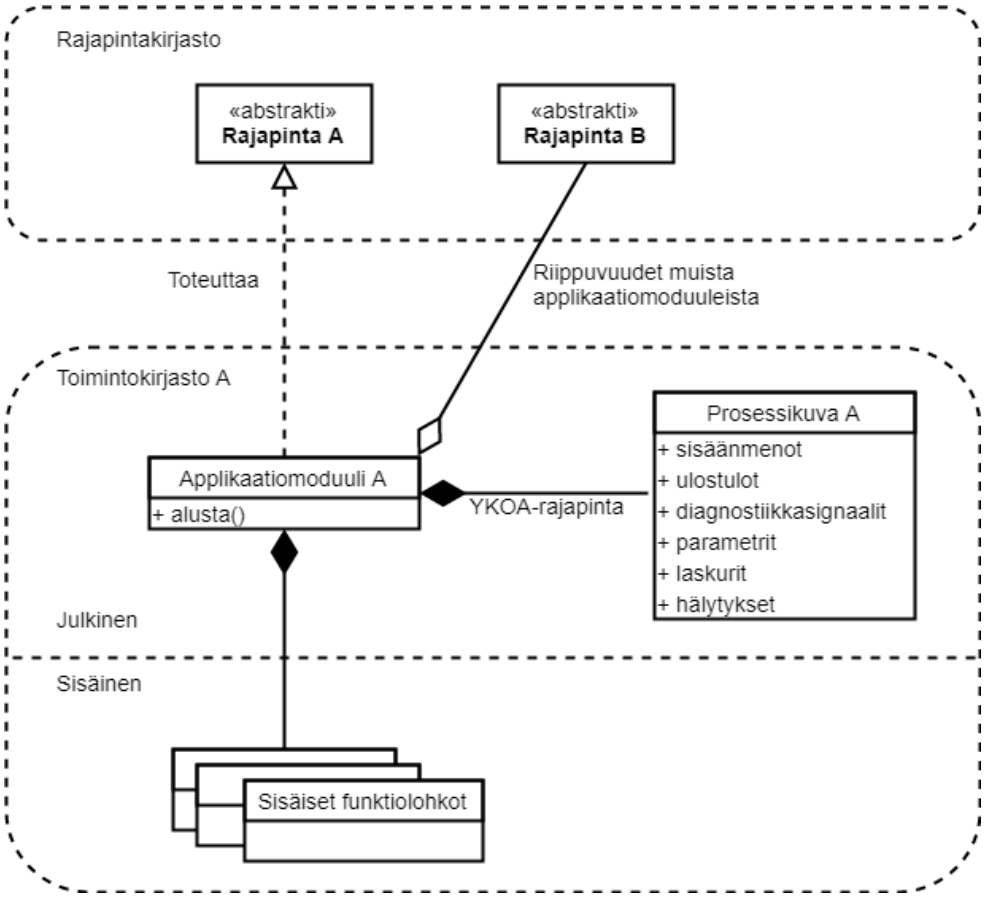


Kuva 19. *UKKO:n CODESYS-kirjastojen hierarkia.*

Toimintokirjastot sisältävät vähintään yhden niin kutsutun applikaatiomoduulin, eli koneenohjauslogiikkaa sisältävän funktiolohkon. Applikaatiomoduuli toteuttaa tyypillisesti

vähintään yhden rajapintakirjastossa määritellyistä abstrakteista rajapinnoista. Kullakin applikaatiomoduulilla on oma prosessikuvansa, joka on osajoukko BASE-applikaation prosessikuvasta. Aivan kuten applikaation prosessikuva, applikaatiomoduulin prosessikuva koostuu sisäänmeno- ja ulostulosignaaleista, diagnostiikkasignaaleista, laskureista, parametreista ja hälytyksistä. Toteutetut rajapinnat ja prosessikuva yhdessä muodostavat applikaatiomoduulin ulkoisen rajapinnan. Muut applikaatiomoduulit käyttävät applikaatiomoduulin palveluita abstraktien rajapintojen kautta, ja applikaatiomoduuli itse vuorovaikuttaa YKOA:n kanssa prosessikuvansa kautta.

Applikaatiomoduuli voi koostua mielivaltaisesta määrästä muita funktiolohkoja. Kapseloinnin periaatteen mukaisesti näiden funktiolohkojen näkyvyysalue on rajattu vain applikaatiomoduulin sisältävän kirjaston sisälle. Kapselointi mahdollistaa applikaatiomoduulin sisäisen toteutuksen muuttamisen järjestelmän ylläpitovaiheessa. Yksittäisen applikaatiomoduulin rakenne on esitetty kuvassa 20.



Kuva 20. *Applikaatiomoduulin koostumus.*

Konemallikohtainen CODESYS-projekti sisältää sovelluksen pääohjelman, joka koostuu kahdesta osasta: alustusosasta, joka suoritetaan kerran heti käynnistyksen jälkeen, ja logiikkaosasta, jota suoritetaan toistuvasti koneen sammuttamiseen asti. Alustusosassa suoritetaan applikaatiomoduulien alustusmetodit, joissa applikaatiomoduulin prosessikuva

sidotaan applikaation prosessikuvaan, ja jossa applikaatiomoduulille syötetään sen tarvitsemat viitteet muihin moduuleihin. Logiikkaosassa kutakin applikaatiomoduulia suoritetaan kerran, jolloin ne ohjaavat konetta. Ideaalitapauksessa konemallikohtaista koodia ei tarvita tämän enempää, ja kaikki koneen ominaisuudet voidaan toteuttaa prosessikuvan parametreilla konemallille sopivaksi säädetyillä UKKO:n valmiilla applikaatiomoduuleilla.

5.5.3 Löysät riippuvuussuhteet ja riippuvuuksien rajoittaminen

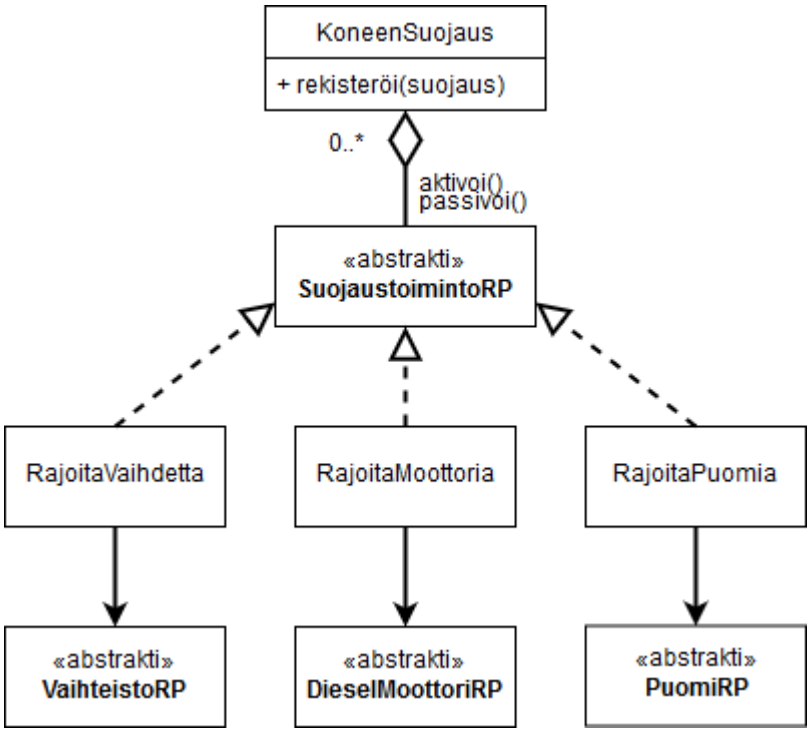
Jos ohjelmistokomponentti on täysin itsenäinen, eli se ei riipu yhdestäkään muusta komponentista, sen uudelleenkäyttö on helppoa. Valitettavasti itsenäisiä komponentteja on vähän, ja ne ovat tyypillisesti yleiskäyttöisiä apukomponentteja, jotka eivät itsessään tee mitään. Kaikki koneen vähänkin monimutkaisemmat ominaisuudet vaativat koneen eri osien yhteistoimintaa, jolloin ominaisuuden toteuttavan komponentin on oltava riippuvainen toisista komponenteista.

Hyvin suunnitellussa järjestelmässä on noudatettu riippuvuuksien kääntämisen periaatetta, ja komponentit ovat riippuvaisia vain abstraktioista konkreettisten toteutusten sijaan. Periaatteen noudattaminen mahdollistaa yksittäisten komponenttien uudelleenkäytön uudessa järjestelmässä ilman sen alkuperäisiä riippuvuuksia. Vaikka komponenttien väliset riippuvuudet olisikin toteutettu abstraktioiden avulla, jokainen riippuvuussuhde monimutkaistaa komponentin uudelleenkäyttöä. Pahimmassa tapauksessa kaikki komponentin riippuvuudet on toteutettava uudelleen, jos mikään alkuperäisistä toteutuksista ei sovellu uuteen käyttöympäristöön. Näin voisi käydä esimerkiksi ensimmäistä UKKO:oon perustuvaa sähkömoottorilla toimivaa konetta kehitettäessä. Prototyyppikone on dieselkone, ja jotkin sitä varten toteutetut komponentit eivät sovellu käytettäväksi sähkökoneessa. Rajapintojen uudelleentoteuttaminen vaatii työtä, mikä ei kannusta komponentin uudelleenkäyttöön. Tästä syystä komponenttien riippuvuudet muihin komponentteihin on minimoitava.

Esimerkki: Koneen suojaustoiminnot. UKKO:oon perustuvassa järjestelmässä koneelle voidaan määritellä suojaustoimintoja, jotka rajoittavat koneen käyttöä, jos koneessa havaintaan vika. Suojaustoiminnot estävät täysitehoisen työskentelyn, mutta mahdollistavat koneen ajamisen huoltoon tai pois muiden koneiden tieltä. Esimerkkejä prototyyppikoneen suojaustoiminnoista ovat moottorin tehon rajoittaminen, suurimman sallitun vaihteen rajoittaminen ja puomin liikkeiden maksiminopeiden rajoittaminen. Laukaisevia vikoja suojaustoiminnoille voivat olla muun muassa moottorin ylikuumeneminen ja keskusvoitelujärjestelmän voiteluaineen käyminen vähiin.

Suojaustoimintojen toteutus on esitetty kuvassa 21. Käytössä olevat suojaustoiminnot ja niiden aktivoitumisehdot voivat vaihdella eri konemallien välillä ja asiakkaan toiveiden mukaan. UKKO tarjoaa siksi yleiskäyttöisen komponentin (KoneenSuojaus), joka valvoo

järjestelmän hälytyksiä, tarkistaa suojaustoimintojen ehdot omista konfiguraatioparametreistaan, ja tarvittaessa aktivoi suojaustoiminnot. KoneenSuojaus-komponentille voidaan applikaation alustuksessa rekisteröidä lähes mielivaltainen määrä suojaustoimintoja.



Kuva 21. *Koneensuojaustoimintojen toteutus.*

Oleellinen asia huomata kuvan 21 ratkaisusta on, että vaikka koneen suojaustoiminnot vaikuttavat applikaatiomoduulien (vaihteisto, moottori, puomi jne.) toimintaan, KoneenSuojaus-komponentti ei ole riippuvainen niistä, eikä toisin päin. Sen sijaan KoneenSuojaus rajoittaa applikaatiomoduulien toimintaa minimaalisen SuojaustoimintoRP-rajapinnan kautta. Rajapinnan toteuttava komponentti vastaa rajoituksen toteuttamisesta käyttäen rajoitettavien applikaatiomoduulien rajapintoja.

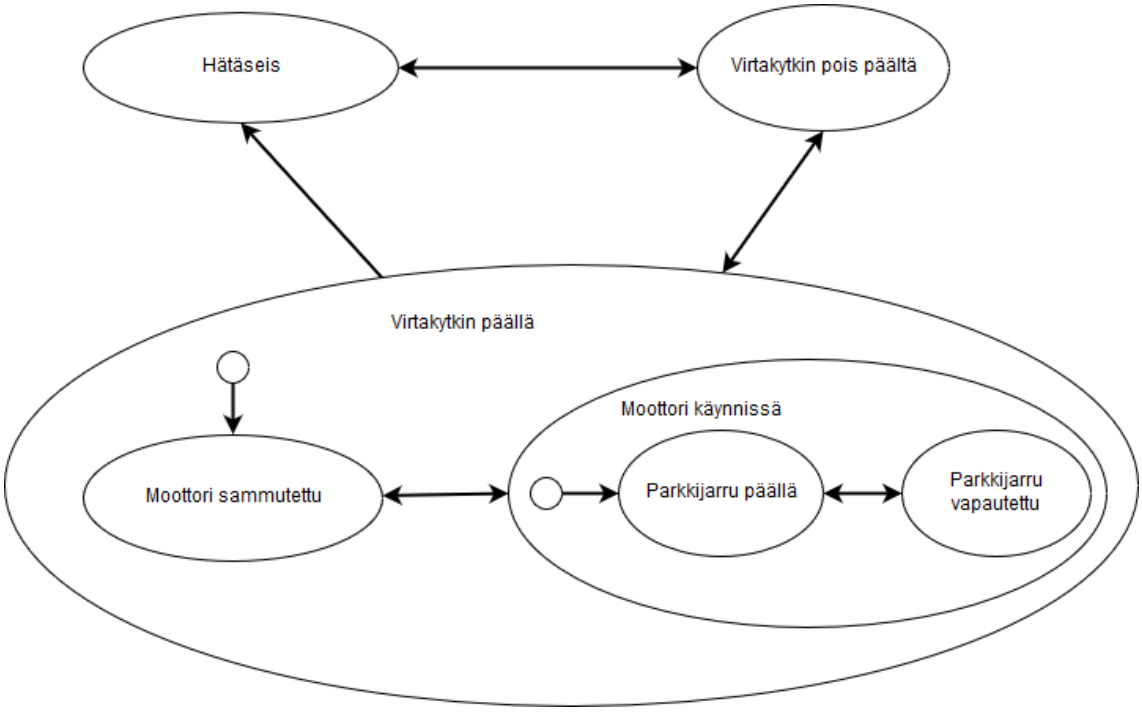
Jos KoneenSuojaus-komponentti joutuisi itse toteuttamaan suojaustoiminnot suoraan applikaatiomoduulien rajapintojen avulla, se tulisi riippuvaiseksi kaikista rajoitettavista applikaatiomoduuleista, mikä vaikeuttaisi sen uudelleenkäyttöä esimerkiksi kuljetuskoneen toteutuksessa. Kuljetuskoneessa ei ole kaikkia samoja applikaatiomoduuleja kuin lastauskoneessa, ja puuttuville riippuvuuksille pitäisi tehdä Liskovin substituutioperiaatetta rikkovat tynkätoteutukset. Lisäksi uusien suojaustoimintojen ja rajoitettavien applikaatiomoduulien lisääminen vaatisi muutoksia KoneenSuojaus-komponenttiin, mikä rikkoisi avoin/suljettu -periaatetta.

Toinen vaihtoehto suojaustoimintojen toteuttamiselle olisi, että applikaatiomoduulit toteuttaisivat SuojaustoimintoRP-rajapinnan, ja rajoittaisivat itse omaa toimintaansa. Ratkaisu rikkoisi yhden vastuualueen periaatetta. Suojaustoimintojen vaatimusten muuttuminen altistaisi itse applikaatiomoduulin regressiolle, eivätkä UKKO:n asiakasprojektit

voisi räätälöidä suojaustoimintoja mielensä mukaan. Jotkin suojaustoiminnot voivat rajoittaa useampaa applikaatiomoduulia, ja joihinkin applikaatiomoduuleihin voi liittyä useita eri suojaustoimintoja. Nämä ominaisuudet ovat yksinkertaisempaa toteuttaa suojaustoimintojen ollessa applikaatiomoduuleista erillisiä. Kuvan 21 ratkaisussa mikä tahansa suojaustoiminto voidaan avoin/suljettu -periaatteen mukaisesti korvata uudella suojaustoiminnolla muuttamatta olemassa olevaa toteutusta, ja kukin suojaustoiminto voi olla riippuvainen niin monesta applikaatiomoduulista kuin sen tarvitsee.

5.5.4 Yleisen tilan hallinta

Vaikka applikaatio koostuu erillisistä applikaatiomoduuleista, joista kutakin ohjataan erikseen, koneella on joitakin yleisiä tiloja, joissa yksittäisten applikaatiomoduulien ohjaus voi olla sallittu tai estetty. Suurin osa näistä yleisistä tiloista on johdettu koneen turvaominaisuuksista, osa taas käytännön rajoitteista. Kuvassa 22 on esitetty koneen ohjauslogiikan yleiset tilat.

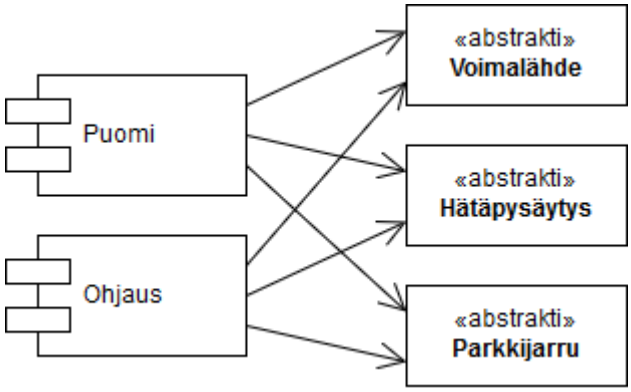


Kuva 22. *Järjestelmän yleiset tilat.*

Hätäseistilassa koneen kaikki liikkeet on estetty, kuten lait ja standardit vaativat. Erillinen turvalogiikka on katkaissut sähköt liikkuvilta osilta, eikä standardipuolen ohjelmisto voi vahingossakaan aiheuttaa koneen liikettä. Standardipuolen ohjelmiston on silti oltava tietoinen hätäseistilasta, jotta järjestelmä pysyy järjissään, ja voi palautua hätäseistilasta pysäytyksen kuittauksen jälkeen. Hätäseistilasta palaudutaan virrattomaan tilaan, jossa koneen liikkeet on edelleen estetty, mutta virrat on mahdollista kytkeä päälle virtakytkimestä kääntämällä.

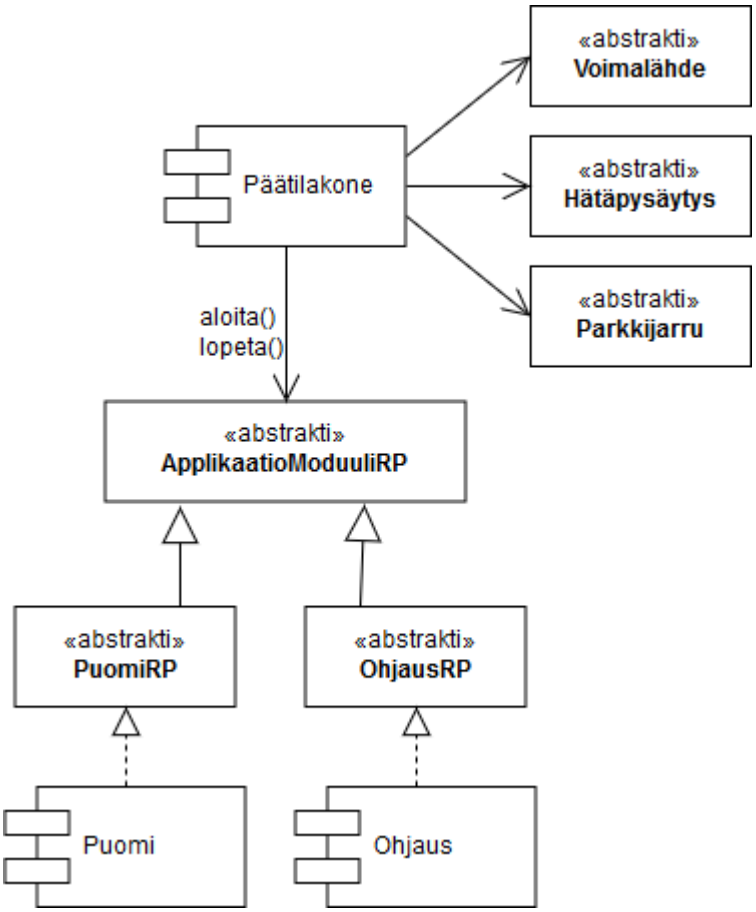
Virtakytkimen päälle kytkemisen jälkeen mahdollistuu ensimmäisten applikaatiomoduulien ohjaus, kuten moottorin käynnistäminen. Koneen muut liikkeet ovat edelleen estettyjä moottorin ollessa sammutettuna, koska ne vaativat voimaa moottorilta. Lisäksi turvalogiikka pitää osan toimilaitteista virrattomina, kunnes parkkijarru on vapautettu. Moottorin käynnistämisen jälkeen parkkijarru on mahdollista vapauttaa, mikä lopulta sallii loppujen applikaatiomoduulien, kuten vaihteiston ja puomin, ohjaamisen. Standardipuolen ohjelmiston näkökulmasta järjestelmässä on vain kuvan mukaiset yleiset tilat. On huomattava kuitenkin, että parkkijarru voi kytkeytyä päälle muustakin syystä kuin operaattorin käskystä. Turvalogiikka voi kytkeä parkkijarrun päälle muun muassa avonaisen hytin oven vuoksi.

Koneen yleisen tilan hallintaan on ainakin kaksi mahdollista ratkaisua. Ensimmäinen ratkaisu on esitetty kuvassa 23. Tässä ratkaisussa kukin applikaatiomoduuli tarkistaa oman ohjauksensa esiehdot moottorin, parkkijarrun ja hätäpysäytyksen rajapintojen kautta. Ratkaisun etuna on, että esiehdot on kapseloitu ohjattavan applikaatiomoduulin sisään, ja jos esiehdot muuttuisivat, muutoksia ohjelman muihin osiin ei tarvita. Huonona puolena on ratkaisusta syntyvän duplikaattikoodin määrä kaikkien applikaatiomoduulien tehdessä lähes samat tarkastelut omalta osaltaan. Applikaatiomoduuleille syntyy ratkaisussa myös kolme uutta riippuvuussuhdetta, jotka vaikeuttavat moduulien uudelleenkäyttöä.



Kuva 23. *Esiehtojen tarkistus: Kaikki moduulit tarkistavat itse esiehtonsa.*

Toinen mahdollinen ratkaisu yleisen tilan hallintaan on esitetty kuvassa 24. Tämä ratkaisu valittiin lopulliseksi toteutustavaksi. Ratkaisussa applikaatiomoduulien itsensä sijasta erillinen päätilakone vastaa kaikkien applikaatiomoduulien esiehtojen tarkastelusta. Päätilakone tarkkailee moottorin, parkkijarrun ja hätäpysäytyksen tilaa, ja ilmoittaa applikaatiomoduuleille yhteisen rajapinnan kautta, milloin ne saavat ja eivät saa liikkua. Ratkaisulla on kaksi etua: Applikaatiomoduuleilla on vähemmän uudelleenkäyttöä vaikeuttavia riippuvuuksia, ja koneen yleisen tilan tarkastelu on keskitetty yhden vastuualueen periaatteen mukaisesti erilliseen moduuliin helpottaen muutosten tekemistä esiehtoihin.



Kuva 24. *Esiehtojen tarkistus: Päätilakone tarkistaa kaikki esiehdot.*

Valitun ratkaisun heikkoutena on se, että päätilakoneen on tunnettava kaikkien applikaatiomoduulien esiehdot. Päätilakone on toki mahdollista suunnitella joustavaksi siten, että yksittäisten applikaatiomoduulien esiehdot ovat helposti muokattavissa tekemättä muutoksia itse tilakoneeseen, vaan sen alustuskoodiin asiakasprojektikohtaisen pääohjelman alustusosassa. Onneksi applikaatiomoduulien esiehdot eivät oletettavasti muutu enää alkuperäisen määrittelyn jälkeen, eivätkä ne vaihtele eri konemallien välillä.

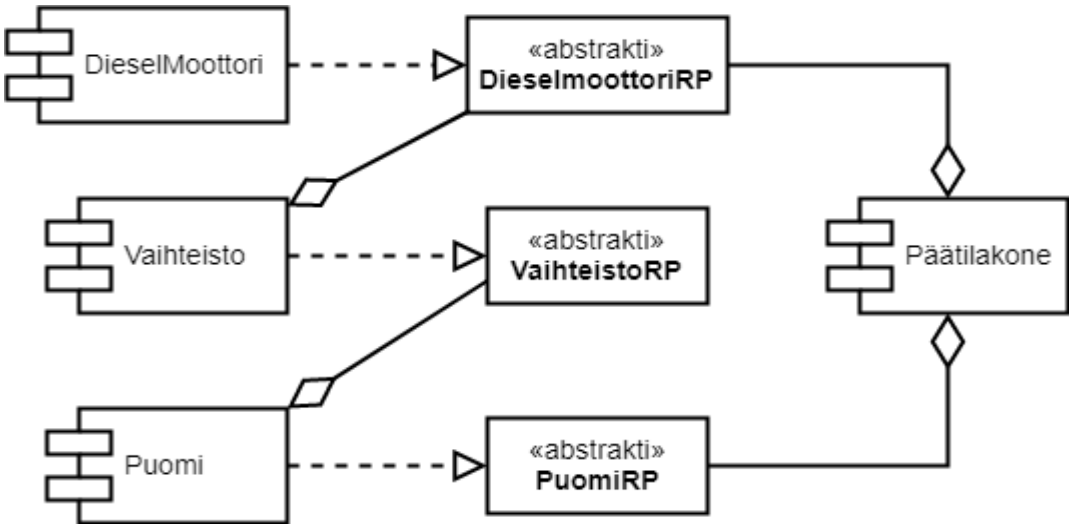
Valitussa ratkaisussa on kuitenkin kiinnitettävä erityistä huomiota uusien applikaatiomoduulien lisäämisen helppouteen. Päätilakone on osa UKKO:n yleisiä kirjastoja, eikä UKKO:n asiakasprojektit voi muokata sitä. On mahdollista, että jotkin projektit haluavat toteuttaa uusia applikaatiomoduuleja ohjaamaan uusia koneen osia, jollaisia nykyisissä koneissa ei ole. Uusien applikaatiomoduulien lisäämisen päätilakoneeseen on oltava avoin/suljettu -periaatteen mukaisesti mahdollista ilman muutoksia päätilakoneen koodiin. Yksikään applikaatiomoduuli ei ole riippuvainen päätilakoneesta, joten asiakasprojektit voivat myös vähäisellä vaivalla toteuttaa kokonaan oman päätilakoneensa, jos valmis ratkaisu ei sovellu heidän tarpeisiinsa.

5.5.5 Rajapintojen eriyttäminen ja korkea abstraktiotaso

Pelkkä abstraktioiden käyttö ja riippuvuuksien kääntämisen periaatteen noudattaminen eivät riitä takaamaan ohjelmiston uudelleenkäytettävyyttä. Huonosti suunnitellut rajapinnat voivat olla loogisesti sidottu yksittäiseen konemalliin, mikä voi estää rajapintaa käyttävien ja niiden toteuttavien moduulien uudelleenkäytön muissa malleissa ilman muutoksia.

Esimerkki: puomin jousituksen toteutus. Puomin jousitus on valinnainen lisäominaisuus, joka parantaa ajomukavuutta ja estää kivien putoamista kuormasta ajon aikana. Jousituksen aktivoituessa puomi nostetaan automaattisesti riittävälle korkeudelle, minkä jälkeen avataan puomin jousitusventtiilit, jotka pehmentävät puomin liikkeitä lastauskoneen ajaessa epätasaisella alustalla. Jousitus on tarkoitettu käytettäväksi vain, kun koneella ajetaan kohtuullista nopeutta, joka on säädettävissä parametrilla. Nopeusmittaus on yksinään epäluotettava, joten ominaisuuden vaatimusmäärittelyssä on lisäksi vaadittu, että koneen absoluuttisen vaihteen on oltava parametrilla säädettävää rajaa suurempi. Tällä tavalla estetään jousitusta aktivoitumasta liian pienellä ajonopeudella epätarkan nopeusmittauksen vuoksi.

Intuitiivinen ratkaisu olisi tarjota kuvan 25 mukaisesti puomille rajapinta (VaihteistoRP), jonka kautta jousituksen toteuttava koodi voi kysyä koneen nykyisen vaihteen. Puomin ohjaus on mahdollista vain moottorin ollessa käynnissä, joten järjestelmässä on oltava myös rajapinta moottorin tilan kysymiselle (DieselMoottoriRP). Kuten edellisessä aliluvussa kerrottiin, riippuvuussuhteiden minimoimiseksi päätilakone vastaa moottorin tilan ja muiden esiehtojen tarkkailusta puomin itsensä sijaan.



Kuva 25. *Intuitiivinen arkkitehtuuri puomin jousituksen toteuttamiseksi.*

Kuvan 25 ratkaisu toimii vain, jos voidaan olettaa, että lastauskoneessa on aina tarjolla toteutukset DieselMoottoriRP- ja VaihteistoRP-rajapinnoille. Näin ei kuitenkaan ole sähkömoottorilla toimivien koneiden tapauksessa. Sähkömoottorilla käyvässä koneessa ei

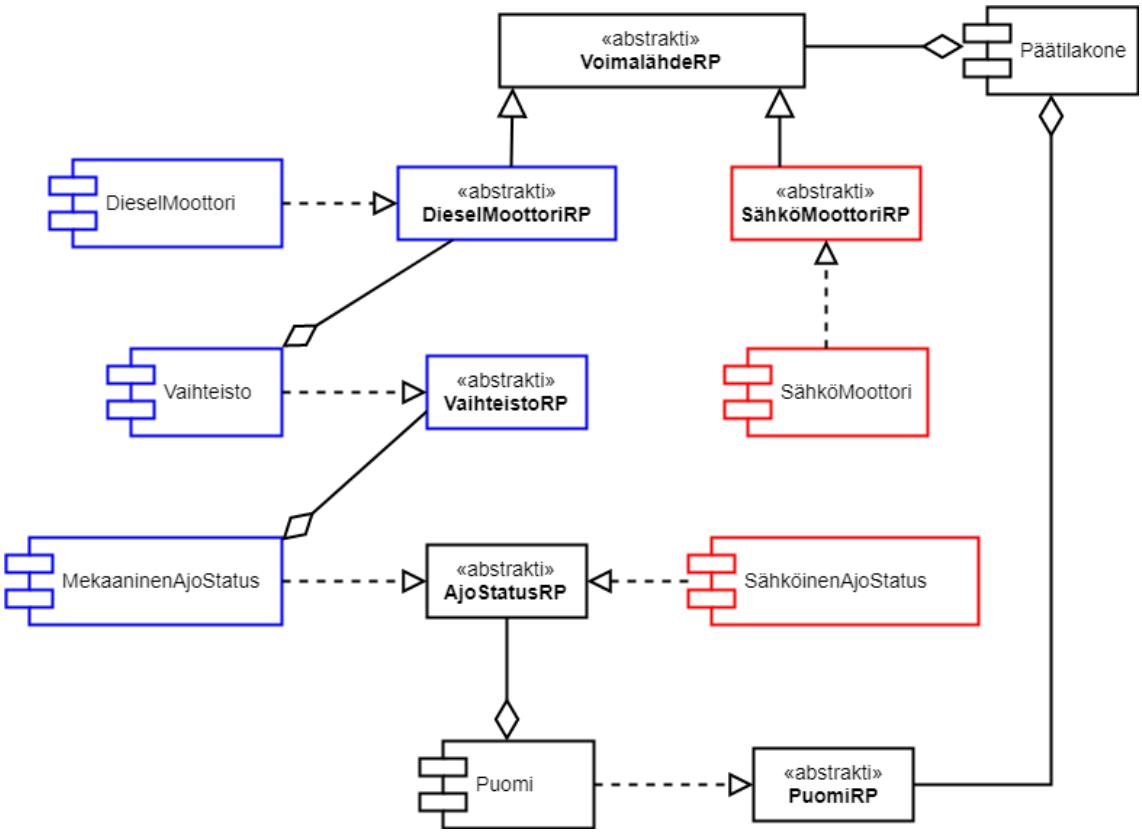
ole samanlaista vaihteistoa kuin dieselkoneissa. Jos Puomi-moduulia haluttaisiin käyttää uudelleen sellaisenaan sähkökoneessa, VaihtoistoRP-rajapinnalle pitäisi tarjota vaihtoehtoinen toteutus. Tämä on vaikeaa, koska vaihdeluvulle tai muille VaihteistoRP-rajapinnan tarjoamille palveluille ei ole mielekkäitä vastineita sähkökoneissa. VaihteistoRP-rajapinnan toteutus sähkökoneessa olisi Liskovin substituutioperiaatetta rikkova tynkä, jonka ainoa tarkoitus on mahdollistaa huolimattomasti suunnitellun Puomi-komponentin uudelleenkäyttö. Tynkätoteutukset aiheuttavat hämmennystä järjestelmän ylläpito- ja jatkokehitysvaiheessa, eikä hyvin suunnitellussa järjestelmässä pitäisi koskaan olla tarvetta niille.

Tynkätoteutukselle vaihtoehtoinen, mutta myös huono ratkaisu olisi tarjota Puomille sekä dieselkoneissa käytettävä VaihteistoRP, että sähkökoneissa käytettävä vastaava rajapinta riittävän ajonopeuden varmistamiseksi. Käytettävä rajapinta valittaisiin suoritusaikana parametrin perusteella. Ratkaisu rikkoisi avoin/suljettu -periaatetta, koska uuden moottorityypin lisääminen vaatisi muutoksia Puomin toteutukseen. Suoritusaikainen rajapinnan valinta pakottaisi kirjoittamaan duplikaattikoodia kullekin valittavalle rajapinnalle ja kasvattaisi bugien mahdollisuutta järjestelmän ylläpito- ja jatkokehitysvaiheessa. Puomi voisi jossakin tilanteessa käyttää vahingossa väärää rajapintaa ja toimia odottamattomalla tavalla.

Myös päätilakoneen riippuvuus DieselMoottoriRP-rajapinnasta osoittaisi huonoa suunnittelua. Dieselmoottorin korvaaminen sähkömoottorilla vaatisi joko muutoksia päätilakoneen toteutukseen, tai että sähkömoottori toteuttaisi DieselMoottoriRP-rajapinnan. Molemmat vaihtoehdot rikkovat olio-ohjelmoinnin SOLID-periaatteita. Muutokset päätilakoneeseen rikkovat avoin/suljettu -periaatetta, koska olemassa olevaa toteutusta joudutaan muuttamaan uuden toiminnallisuuden lisäämiseksi. DieselMoottoriRP-rajapinnan toteuttaminen sähkömoottorin toteutuksessa rikkoo Liskovin substituutioperiaatetta, koska sähkömoottori ei ole dieselmoottori, eikä se voi oikeasti tarjota kaikkia dieselmoottorin rajapinnan palveluita.

Paranneltu arkkitehtuuri on esitetty kuvassa 26. Ongelmien ratkaisemiseksi on tärkeää ymmärtää todelliset motiivit vaatimusmäärittelyjen takana. Puomin jousituksessa vaihdetiedolla haluttiin vain varmistaa, että koneella ollaan ajamassa kohtuullista nopeutta. Vaihdeluku tai nopeus itsessään eivät ole kiinnostavia tietoja jousituksen kannalta. VaihteistoRP-rajapinta voidaan siten puomin toteutuksessa korvata AjoStatusRP-rajapinnalla, joka tarjoaa tiedon koneen ajonopeudesta ja -suunnasta. AjoStatusRP-rajapinta ei sisällä voimansiirtojärjestelmästä riippuvia ominaisuuksia, joten sille voidaan tarjota vaihtoehtoiset toteutukset diesel- ja sähkökoneille ilman tynkätoteutuksia. Rajapinnan toteutus dieselkoneissa (MekaaninenAjoStatus) voi tulkita ajonopeuden ja -suunnan koneen nopeuden ja vaihdeluvun perusteella, kuten puomin jousituksen vaatimusmäärittelyssä vaadittiin. Sähkökoneessa rajapinnan toteutus (SähköinenAjoStatus) voi varmistaa riittävän ajonopeuden jollakin mielekkäämmällä tavalla.

Päätilakoneen ongelma voidaan ratkaista määrittelemällä diesel- ja sähkömoottoreille yhteinen rajapinta (VoimalähdeRP), joka tarjoaa molempien moottorityyppien yhteiset palvelut, kuten tiedon moottorin tilasta. Yhteistä rajapintaa voidaan laajentaa moottorityyppikohtaisilla rajapinnoilla (DieselMoottoriRP ja SähköMoottoriRP). Moottorista riippuvien moduulien tulee käyttää moottoria mahdollisimman korkean tason rajapinnan kautta. Päätilakoneelle riittää tieto moottorin käynnissä olosta, joten se käyttää moottoria yhteisen VoimalähdeRP-rajapinnan kautta. Dieselkoneessa vaihteiston toiminta on aidosti riippuvainen dieselmoottorin ominaisuuksista, joten vaihteiston toteutus käyttää moottoria oikeutetusti laajennetun DieselMoottoriRP-rajapinnan kautta.



Kuva 26. *Paranneltu arkkitehtuuri puomin jousituksen toteuttamiseksi.*

Kuvassa 26 dieselkoneen toteutuksessa käytettävät osat on merkitty sinisellä, sähkökoneen toteutuksessa käytettävät osat punaisella ja yhteiset osat mustalla. Kuvasta näkyy, miten arkkitehtuuri toteuttaa olio-ohjelmoinnin avoin/suljettu -periaatetta. Ensimmäisen sähkömoottorilla toimivan konemallin kehittäminen ei vaadi muutoksia yhteisiin komponentteihin. Yhteisille rajapinnoille tarjotaan vain uudet toteutukset niille osille, jotka riippuvat moottorin tyypistä. Dieselkoneen ja sähkökoneen komponentit ovat toisilleen vaihtoehtoisia. Sähkökoneessa dieselkoneen komponentit voidaan jättää pois sovelluksesta, ja sama toisin päin.

Esimerkin mukaista rajapintojen eriyttämistä ja rajapintojen laajentamista on sovellettu koko järjestelmän suunnittelussa. Pääsääntönä on, että rajapintojen, joista komponentit

riippuvat, tulisi olla mahdollisimman tarkasti rajattu käyttötarkoituksen mukaan. Vaatimusmäärittelyistä on tunnistettava niiden todellinen tarkoitus, ja jos mahdollista, abstrahoitava vaatimuksen toteutus abstraktin, koneen yksityiskohdista riippumattoman rajapinnan taakse. Tällä tavoin komponentit pysyvät mahdollisimman riippumattomina koneen muista osista, jotka voivat estää komponentin uudelleenkäytön erityyppisissä koneissa.

5.5.6 Erilliset ohjausmoodit, yhteinen ohjaus

Kuten aliluvussa 5.2.2 kerrottiin, UKKO mahdollistaa koneen ohjaamisen kolmessa ohjausmoodissa: paikallisesti hytistä käsin, radio-ohjaimella ja automaatio-ohjauksella. Vain yksi näistä voi olla kerrallaan käytössä. Etäohjauksessa osa koneen toiminnoista voi olla estettyjä, mutta koneen perustoiminnot ovat käytössä kaikissa ohjausmoodeissa.

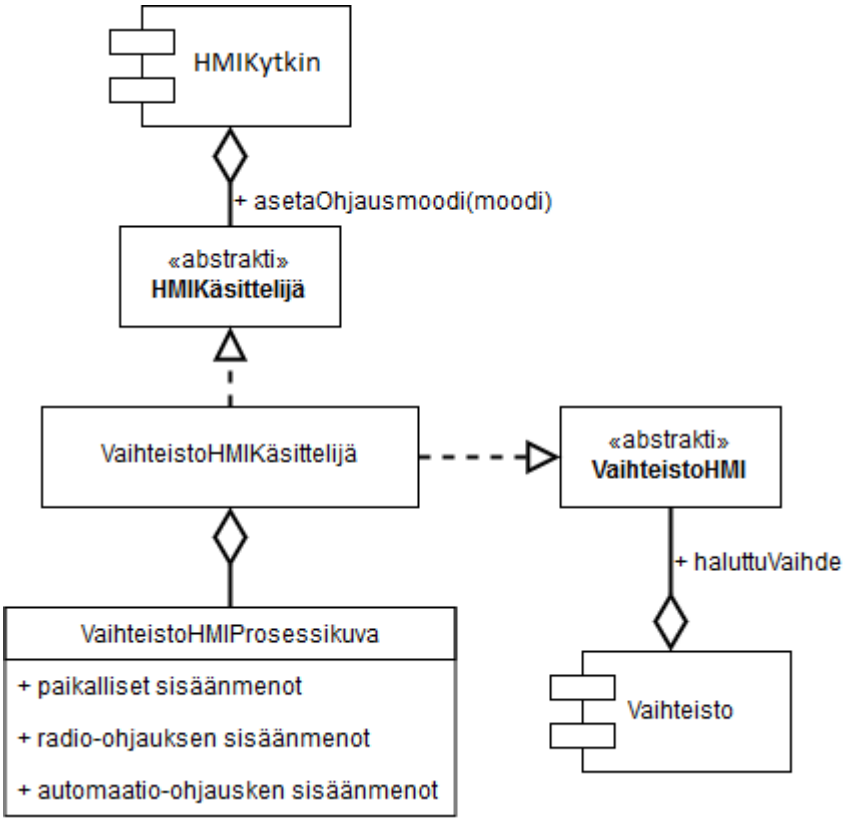
Ohjauskomennot voivat poiketa toisistaan eri moodien välillä. Esimerkiksi prototyyppikoneessa kuljettaja voi hytistä vaihtaa ajosuuntaa ohjaussauvan suuntakytkimellä, ja vaihtaa vaihdetta vaihde kerrallaan ylös tai alas ohjaussauvan painikkeilla. Automaatio-ohjauksessa haluttu vaihde vastaanotetaan Ethernet-väylältä kokonaislukuna, jossa negatiiviset arvot tarkoittavat taaksepäin suuntaa. Radio-ohjauksessa on käytössä vain ensimmäinen vaihde molempiin suuntiin. Suunta päätellään CAN-väylältä vastaanotetusta viestistä, joka ilmaisee radio-ohjaimen ajonopeusohjaussauvan asentoa väliltä -100 – 100 %. Vaikka ohjauskomentojen logiikka olisikin täsmälleen sama kaikissa ohjausmoodeissa, kullakin ohjausmoodin komennolla on oma sisäänmenosignaalinsa applikaation prosessikuvassa. Jokaiselle ohjauskomennolle on siten vähintään kolme sisäänmenoa. Tämä koskee kaikkien applikaatiomoduulien kaikkia ohjausviestejä.

Riippumatta valitusta ohjausmoodista koneen on vastattava komentoihin samalla tavalla. Jos komentojen sisäänmenosignaalit käsiteltäisiin niiden ohjaamassa applikaatiomoduulissa, toteutuksesta tulee helposti sotkuinen. Joka kerta kun ohjauslogiikan on tarkistettava komentosignaalin arvo, sen on tarkistettava myös nykyinen ohjausmoodi ja tulkittava oikeat sisäänmenosignaalit sen mukaan. Lisäksi koneen ohjaimet ja siten komentojen sisäänmenosignaalit voivat vaihdella eri mallien välillä. Ohjauslogiikan ja komentojen tulkinnan yhdistäminen rikkoisi yhden vastuualueen periaatetta ja estäisi ohjauslogiikan uudelleenkäytön koneessa, jossa ohjaimet poikkeavat vähänkään prototyyppikoneesta.

Siksi on järkevää eriyttää ohjauslogiikka komentojen tulkitsemisesta. UKKO:n ratkaisu ohjauslogiikan ja komentojen käsittelyn eriyttämiseen on esitetty kuvassa 27. UKKO tarjoaa applikaatiomoduulin (HMIKytkin), jonka tehtävänä on valita turvalogiikan ilmoittaman ohjausmoodin perusteella oikeat ohjauskomennot ja välittää ne applikaatiomoduuleille ohjausmoodeista riippumattomassa muodossa.

HMIKytkin koostuu käsittelijöistä, joista kukin vastaa yksittäisen applikaatiomoduulin komentojen tulkitsemisesta. HMIKytkin kertoo käsittelijöille nykyisen ohjausmoodin

abstraktin HMIKäsittelijä-rajapinnan kautta. HMIKäsittelijä-rajapinta on yhteinen kaikkien applikaatiomoduulien komentojen käsittelijöille. HMIKytkin ei ole muutenkaan riippuvainen ohjattavista applikaatiomoduuleista, minkä vuoksi samaa komponenttia voi käyttää sellaisenaan uudelleen kuljetuskoneen toteutuksessa, vaikka siinä on käytössä eri applikaatiomoduulit kuin lastauskoneessa. Käsittelijöiden määrää tai konkreettisia tyyppejä ei ole kovakoodattu, vaan lista käsittelijöistä injektoidaan HMIKäsittelijä-rajapinnan viitteinä HMIKytkimen alustuksessa. Uusien käsittelijöiden lisääminen tai UKKO:n tarjoamien oletuskäsittelijöiden vaihtaminen onnistuu siten avoin/suljettu -periaatteen mukaisesti ilman muutoksia olemassa oleviin komponentteihin.



Kuva 27. *Eri ohjausmoodien komentojen yhtenäistäminen.*

Kukin käsittelijä toteuttaa myös ohjattavalle applikaatiomoduulille erityisen HMI-rajapinnan, jonka kautta applikaatiomoduuli voi kysyä nykyisen ohjausmoodin komennot. HMI-rajapinnan ansiosta ohjauslogiikan toteuttavan applikaatiomoduulin ei tarvitse olla tietoinen valitusta ohjausmoodista, komentojen sisäänmenosignaaleista tai niiden tulkinnasta. Kuvan 27 esimerkissä vaihteistoa ohjaava applikaatiomoduuli saa operaattorin haluaman vaihteen VaihteistoHMI-rajapinnan kautta. Rajapinta tarjoaa halutun vaihteen ja suunnan yhtenä kokonaislukuna riippumatta ohjausmoodista. Automaatio-ohjauksessa Ethernet-väylältä vastaanotettu vaihdepyynti kelpaa sellaisenaan rajapinnan ilmoittamaksi arvoksi. Radio-ohjauksessa ja paikallisessa ohjauksessa HMI-rajapinnan ilmoittama arvo on jalostettava todellisista, käsittelijän omassa prosessikuvassa listatuista sisäänmenosignaaleista.

Käsittelijän prosessikuva on osajoukko applikaation prosessikuvasta ja alustetaan applikaation alustusvaiheessa, aivan kuten applikaatiomoduulien prosessikuvat. Prosessikuva on heikoin lenkki HMI-käsittelijän uudelleenkäytön kannalta. Prosessikuva on mahdollista vakioida radio-ohjauksen ja automaatio-ohjauksen osalta, mutta paikallisen ohjauksen sisäänmenosignaalit voivat hyvinkin vaihdella konetyyppien välillä. Paikallisen ohjauksen sisäänmenosignaalit perustuvat koneen sähköisiin kytkentöihin. Väyläabstraktion ansiosta ohjauslaitteiden sijoittelulla ei ole merkitystä, mutta jos ohjauslaitteen tuottama signaali on erilainen kuin alkuperäisessä prototyypissä, HMI-käsittelijä on toteutettava uudelleen. Esimerkiksi jos uudessa konemallissa vaihteen valinta tehtäisiin ohjaussauvan ylös - ja alas painikkeiden sijaan yhdellä valintakytkimellä, prototyyppikonetta varten tehty HMI-käsittelijä ei soveltuisi uuteen malliin. Onneksi HMI-käsittelijät ovat yksinkertaisia komponentteja, joten niiden toteuttaminen uudelleen ei ole kohtuuttoman työlästä.

6. YHTEENVETO

Tässä diplomityössä käsiteltiin ohjelmiston uudelleenkäyttöä koneenohjausjärjestelmien ja etenkin liikkuvien kaivostyökoneiden sovellusalalla. Koneenohjausjärjestelmällä tarkoitetaan laitteistoa ja ohjelmistoa, joka saa koneen tekemään työnsä turvallisesti operaattorien komentojen mukaisesti.

Koneenohjausjärjestelmien ohjelmisto on laitteistoläheistä, ja usein hajautettu useille eri ohjausyksiköille. Ohjausyksikön IO-rajapinta voi koostua eri tyyppisistä sisäänmenoista ja ulostuloista. IO-rajapinnan käyttö on riippuvainen ohjausyksikköön liitetyistä antureista ja toimilaitteista, jotka voivat vaihtua koneen elinkaaren aikana ja uusia malleja kehitettäessä. Ohjausyksiköt yleensä kommunikoivat keskenään kommunikaatioväylien avulla yksi usealle -mallin mukaisesti. Väylätoteutuksia on olemassa useita, ja kullakin toteutuksella voi olla useita vaihtoehtoisia korkean tason protokollia. Järjestelmän väylärakenne voi vaihdella kehitystyön aikana ja eri konemallien välillä. Vaihtelevuus järjestelmän IO-rajapinnassa ja väylärakenteessa, räätälöitävät ominaisuudet ja kehittyvä turvallisuuslainsäädäntö vaikeuttavat ohjelmiston uudelleenkäyttöä.

Abstraktiot mahdollistavat uudelleenkäytettävyyden toteuttamisen. Laitteistoabstraktio auttaa rajaamaan IO-rajapinnan ja muun laitteiston muutoksista aiheutuvat ohjelmistomuutokset ohueen abstraktiokerrokseen. Väyläabstraktio vastaavasti suojaa sovelluslogiikkaa väylärakenteen muutoksilta. Yhdistämällä laitteisto- ja väyläabstraktio yhteisen IO-abstraktion avulla sovelluslogiikka voi olla täysin riippumaton laitteiston yksityiskohdista ja väylärakenteesta. Huolellisesti suunniteltu IO-abstraktio on täysin konfiguroitava, eivätkä muutokset järjestelmän laitteistossa ja kytkennöissä vaadi muutoksia itse ohjelmistoon.

Olio-ohjelmointi helpottaa uudelleenkäytettävän ohjelmiston suunnittelua ja toteutusta. Koneenohjauksessa käytettävät olio-ohjelmointikielet tarjoavat perinteisiä proseduraalisia kieliä enemmän työkaluja abstraktioiden toteuttamiseen, mutta sallivat riittävän laitteistoläheisyyden toisin kuin modernit funktionaaliset kielet. Olio-ohjelmoinnin SOLID-periaatteita noudattamalla ohjelmisto on mahdollista suunnitella modulaariseksi, laajennettavaksi, erilaistettavaksi ja helposti ymmärrettäväksi, ylläpidettäväksi ja jatkokehitettäväksi.

Diplomityössä esiteltiin Sandvikin UKKO-koneenohjausjärjestelmää käytännön esimerkkinä uudelleenkäytettäväksi tarkoitetusta koneenohjausjärjestelmästä. UKKO on suunniteltu ja toteutettu laitteisto- ja väyläabstraktioita käyttäen ja olio-ohjelmoinnin SOLID-periaatteita soveltaen. Diplomityössä osoitettiin esimerkein, miten koneen ominaisuudet ovat muunneltavissa ja osat vaihdettavissa ilman muutoksia olemassa olevaan ohjelmistoon. Järjestelmä vaikuttaa siten täyttävän lupauksensa uudelleenkäytettävyydestä.

Diplomityön julkaisuhetkellä UKKO:n kehitystyö on vielä kesken. Vaikka järjestelmä on suunniteltu uudelleenkäytettäväksi eri konemallien välillä parhaan insinööritaidon mukaan ja yleisesti hyviksi todettuja periaatteita noudattaen, todellinen näyttö tavoitteiden saavuttamisesta saadaan vasta ensimmäisen erilaisen koneen kehitystyön myötä. Tällä hetkellä tavoitteet näyttävät olevan saavutettavissa.

LÄHTEET

[1] World's most automated underground mine, http://www.northparkes.com/improvement/worlds-most-automated-underground-mine, viitattu 20.1.2018

[2] PLC Mentor, History of the Programmable Logic Controller (PLC), http://www.plcmentor.com/Articles/Newsletters/Programmable-Logic-Controller-PLC-History.aspx, viitattu 2.6.2016.

[3] W. Bolton, Programmable Logic Controllers, 5th ed. Newnes, Burlington, 2011.

[4] D.A. Mindell, Johns Hopkins Studies in the History of Technology: Between Human and Machine: Feedback, Control, and Computing before Cybernetics, Johns Hopkins University Press, 2002, s. 276-306.

[5] Epec Oy, Epec 3724 Technical Document, saatavilla: http://bram-engineers.nl/wp-content/uploads/2013/12/3724_MAN000553.pdf, viitattu 3.6.2018.

[6] CiA 301 v. 4.2.0 CANopen application layer and communication profile, CAN in Automation CiA, 2011, saatavilla: https://www.can-cia.org/standardization/specifications/

[7] V. Eloranta, J. Koskinen, M. Leppänen, V. Reijonen, Designing Distributed Control Systems: A Pattern Language Approach, John Wiley & Sons, Chichester, 2014.

[8] J.J. Hänninen, Kaivoslastauskoneiden CANopen-ohjausjärjestelmän simulointisovellus, 2016, Saatavilla: https://dspace.cc.tut.fi/dpub/handle/123456789/23675

[9] CiA CANopen history, https://www.can-cia.org/can-knowledge/canopen/canopen-history/.

[10] Euroopan parlamentin ja neuvoston direktiivi 2006/42/EY, http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32006L0042:FI:HTML#ntr13-L_2006157FI.01002401-E0013.

[11] Valtioneuvoston asetus koneiden turvallisuudesta 12.6.2008/400, http://plus.edilex.fi/tukes/fi/lainsaadanto/20080400.

[12] G. O'Regan, History of Programming Languages. kirjasta: A Brief History of Computing. Springer, London, 2012, s. 121-144.

[13] M. Banahan, D. Brandy, M. Doran, The C Book, Addison Wesley, 1991, saatavilla: http://publications.gbdirect.co.uk/c_book/

[14] The C standard wiki page, http://www.iso-9899.info/wiki/The_Standard, viitattu 16.3.2018.

[15] History of C++, cplusplus.com, http://www.cplusplus.com/info/history/, viitattu 16.3.2018.

[16] C++ Standards, isocpp,org, https://isocpp.org/std/the-standard, viitattu 16.3.2018.

[17] PLCopen.com, TC1 – Standards, http://www.plcopen.org/pages/tc1_standards/, viitattu 8.6.2018.

[18] 3S Smart Software Solutions, CODESYS, https://www.codesys.com/the-system.html, viitattu 8.6.2018

[19] K. John, M. Tiegelkamp, IEC 61131-3: programming industrial automation systems, 2. ed. ; [softcover reprint of the hardcover 2. ed. 2010] ed. Springer, Berlin, 2014.

[20] M. Rintala, J. Savela, Olioiden ohjelmointi C++:lla, 2013, s. 26-53 saatavilla: http://www.cs.tut.fi/~oliot/kirja/olioiden-ohjelmointi-uusin.pdf

[21] G. Fischer, Cognitive View of Reuse and Redesign, IEEE Software, Vol. 4, Iss. 4, 1987, s. 60-72.

[22] R.L. Biddle, E.D. Tempero, Inheritance and reusability, Proceedings 1998 Australian Software Engineering Conference (Cat. No.98EX233), pp. 184-191.

[23] S. Roth, Clean C++ : Sustainable Software Development Patterns and Best Practices with C++ 17, Apress L. P, Berkeley, CA, 2017.

[24] V. Varjoranta, Software safety issues in machine control system design processes, 2012, saatavilla: https://dspace.cc.tut.fi/dpub/handle/123456789/21028.

[25] W. Ecker, R. Dömer, W. Müller, Hardware-Dependent Software: Principles and Practice, Springer, Dordrecht, 2009.

[26] W.B. Frakes, S. Isoda, Success factors of systematic reuse, IEEE Software, Vol. 11, Iss. 5, 1994, s. 14-19.

[27] J. Long, Software Reuse Antipatterns-Revisited, Software Quality Professional, Vol. 19, Iss. 4, 2017

[28] L.D. Spencer, S.H. Richards, Reliable JavaScript: How to Code Safely in the World's most Dangerous Language, John Wiley & Sons, Inc, Hoboken, NJ, USA, 2015, s. 1-27.

[29] W. Haoyu, Z. Haili, Basic Design Principles in Software Engineering, 2012 Fourth International Conference on Computational and Information Sciences, IEEE, s. 1251-1254.

[30] ISO 11898-1:2015, saatavilla: https://www.iso.org/standard/63648.html.

[31] ISO 11898-2:2016, saatavilla: https://www.iso.org/standard/67244.html

[32] ISO 11898-3:2006, saatavilla: https://www.iso.org/standard/36055.html

[33] ISO 11898-4:2004, saatavilla: https://www.iso.org/standard/36306.html

[34] ISO 11898-5:2007, saatavilla: https://www.iso.org/standard/41284.html

[35] ISO 11898-6:2013, saatavilla: https://www.iso.org/standard/59165.html

[36] About CANopen, http://www.canopensolutions.com/english/about_canopen/about_canopen.shtml, viitattu 25.5.2018.

[37] M. Di Natale, H. Zeng, P. Giusto, A. Ghosal, I. Books24x7, Understanding and Using the Controller Area Network Communication Protocol : Theory and Practice, Springer, New York, NY, 2012.

[38] CAN in Automation kotisivut, https://www.can-cia.org, viitattu 26.5.2018.

[39] SAE International kotisivut, https://www.sae.org, viitattu 26.5.2018.

[40] ASAM kotisivut, https://www.asam.net, viitattu 26.5.2018.

[41] ODVA kotisivut, https://www.odva.org/, viitattu 26.5.2018.

[42] ISO 11783-1:2017, saatavilla: https://www.iso.org/standard/57556.html

[43] Node Monitoring via Node-Guarding or Heartbeat Messages, CANopen Solutions, http://www.canopensolutions.com/english/about_canopen/guarding_heartbeat.shtml, viitattu 28.5.2018.

[44] M. Junger, Introduction to J1939, Vector, 2010, saatavilla: https://vector.com/portal/medien/cmc/application_notes/AN-ION-1-3100_Introduction_to_J1939.pdf

[45] EN 1889-1:2011, saatavilla: https://shop.bsigroup.com/ProductDetail/?pid=000000000030185767&industry=mining, viitattu 4.6.2018

[46] EN ISO 4413:2010, saatavilla: https://www.iso.org/standard/44781.html

[47] EN 60204-1:2006, saatavilla: https://shop.bsigroup.com/ProductDetail/?pid=000000000030218354

[48] EN 60529:1992, saatavilla: https://shop.bsigroup.com/ProductDetail/?pid=000000000030214185

[49] EN ISO 3411:2007, saatavilla: https://www.iso.org/standard/38911.html

[50] EN ISO 3449:2005, saatavilla: https://www.iso.org/standard/32900.html

[51] EN ISO 3450:2008, saatavilla: https://www.iso.org/standard/42076.html

[52] EN ISO 3471:2008, saatavilla: https://www.iso.org/standard/38084.html

[53] EN ISO 6682:1986, saatavilla: https://www.iso.org/standard/13111.html

[54] EN ISO 12100:2010, saatavilla: https://www.iso.org/standard/51528.html

[55] EN ISO 13849-1:2006, saatavilla: https://www.iso.org/standard/34931.html

[56] ISO 5010:2007, saatavilla: https://www.iso.org/standard/45105.html

[57] EN ISO 13850, saatavilla: https://www.iso.org/standard/59970.html

[58] EN 62061, saatavilla: https://shop.bsigroup.com/ProductDetail/?pid=000000000030313541

[59] EN 60947-1:2007, saatavilla: https://shop.bsigroup.com/ProductDetail/?pid=000000000030249098

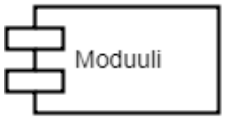
[60] AS 4024, saatavilla: https://infostore.saiglobal.com/en-us/standards/as-4024-1-2014-series-1771762/

[61] CAN/CSA-M424.2, saatavilla: http://shop.csa.ca/en/canada/mine-safety-standards/m4242-16/invt/27006752016

[62] IEC 61508, saatavilla: http://www.iec.ch/functionalsafety/

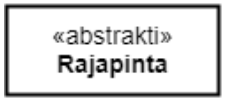
LIITE A: ESIMERKEISSÄ KÄYTETYT UML-MERKINNÄT

Olio, eli luokan tai funktiolohkon instanssi

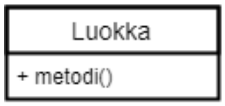


Ohjelmistomoduuli. Tässä diplomityössä käytetty erityisesti kuvaamaan applikaatiomoduuleja.

Kirjasto. Sisältää kokoelman applikaatiomoduuleja tai muita funktiolohkoja.

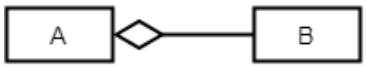


Abstrakti rajapinta.

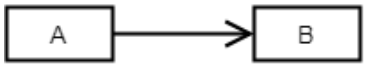


Luokka / funktiolohko.

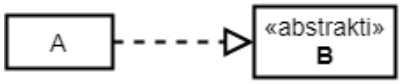
Komposiitio, A koostuu B:stä.



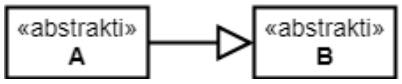
Aggregaatio, A sisältää viitteen B:hen.



Assosiaatio, A riippuu jollakin tapaa B:stä.



Rajapinnan toteutus, A toteuttaa rajapinnan B.



Rajapinnan laajentaminen, Rajapinta A laajentaa rajapintaa B.